Performance Analysis and Prediction of Large-Scale Scientific Applications

Adolfy Hoisie and Harvey Wasserman

{hoisie, hjw}@lanl.gov

Computer & Computational Sciences Division (CCS)

Computer, Communications, and Networking Division (CCN)

University of California

Los Alamos National Laboratory

Los Alamos, New Mexico 87545

ASCI PI January 8, 2002

Introduction

- This talk represents a drastically-reduced version of a tutorial given by the authors at SC99, SC00, and elsewhere.
- Because of the time constraints, no one subject in the tutorial can be discussed completely.
- The talk is basically divided into two portions:
  - Single-processor performance issues
  - Multiprocessor performance modeling

Full Tutorial Outline From SC2000

Introduction, definitions, short overview of performance analysis techniques	15 minutes
Performance metrics – serial	45 minutes
Performance optimization for serial applications – an overview of specific techniques	30 minutes
COFFEE BREAK !!!	30 minutes
Modeling and measurement of memory performance	45 minutes
Performance metrics – parallel	45 minutes
LUNCH BREAK!!!	90 minutes
Performance optimization for parallel applications – an overview of specific techniques	30 minutes
COOKIE BREAK will occur some time in case studies	30 minutes
Scalability Analysis Case Studies	90 minutes
Performance prediction on future systems	30 minutes
Conclusions, lessons learned, wrap-up	15 minutes

See http://www.c3.lanl.gov/par_arch

Motivation

" 20% of a project's time is spent in trying to understand what to build, 80% is spent building it, and no time is spent trying to understand deeply, how well the design decisions were made in terms of performance delivered to users, and hence, how to proceed on the next system design."

- David Kuck,
Kuck & Associates, Inc. and
Univ. of Illinois, Emeritus
"High-Performance Computing"
Oxford U. Press, 1996

Single-Processor Performance Issues

"…it is now obvious that no one knows how to deliver practical parallel systems…"

- ***David Kuck,***
Kuck & Associates, Inc. and
Univ. of Illinois, Emeritus
"High-Performance Computing"
Oxford U. Press, 1996

"Fundamental Equation" of Performance Modeling

$$T_{run} = T_{computation} + T_{communication} - T_{overlap}$$

- **$T_{computation}$** A coarse approximation is based on the number of FLOPS per grid point and a characteristic MFLOP rate.
- **$T_{communication}$** is tricky. It depends on the type of communication (blocking, non-blocking), communication pattern, HW communication parameters, network topology, and contention. The linear model (latency-bandwidth) or LogGP can be utilized.
- **$T_{overlap}$** is the hardest. It depends on algorithmic overlap, communication /computation overlap in hardware, load balancing, contention, runtime variability, overall machine load.

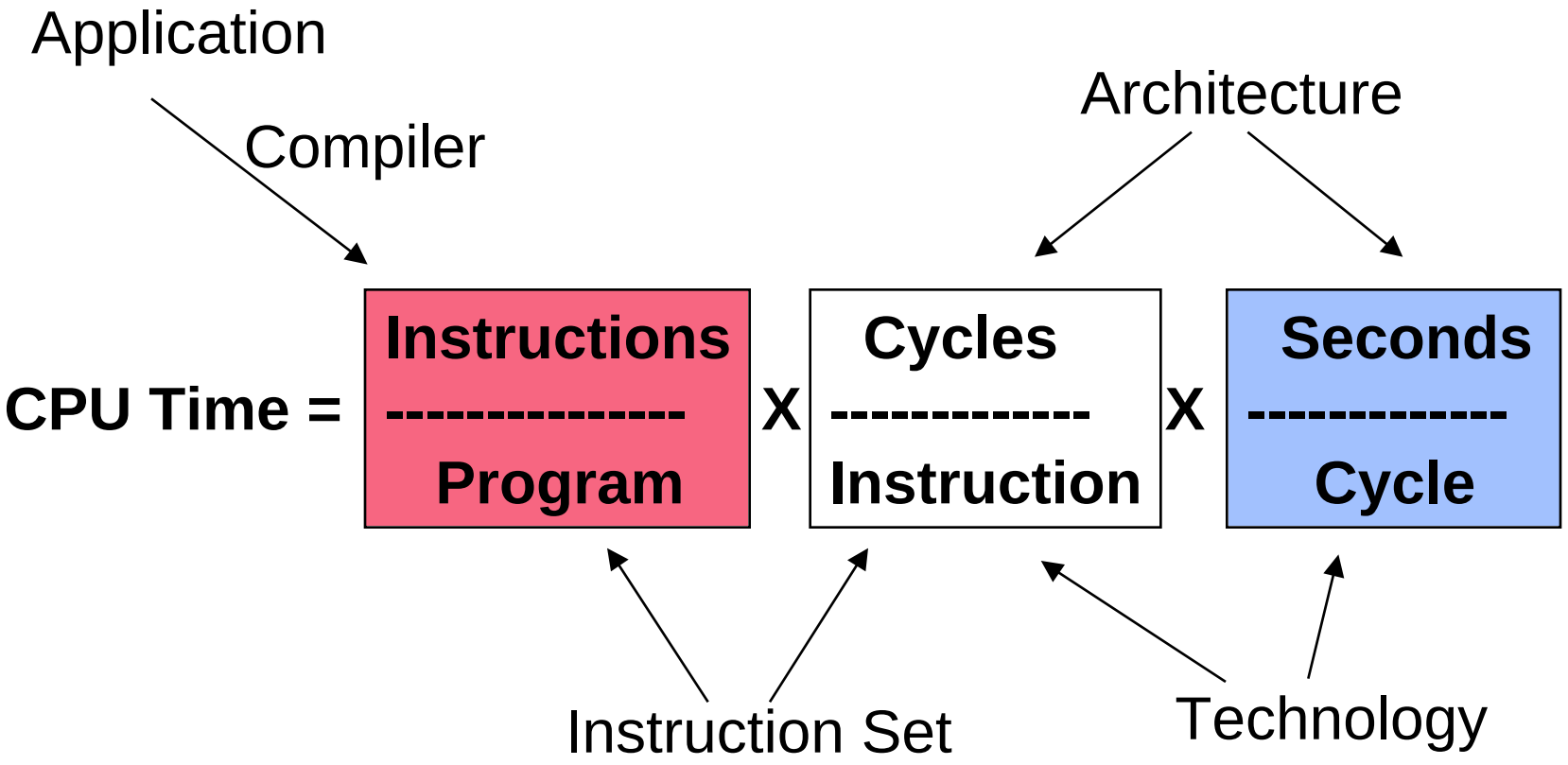
Serial Performance

- Rate = ***f*** * Peak_Rate

f ≡ 0.333333 sPPM Only

f = ~0.1 ± .05 Everything else

Aspects of Serial Performance



CPU Time = N_{inst} * CPI * Clock rate

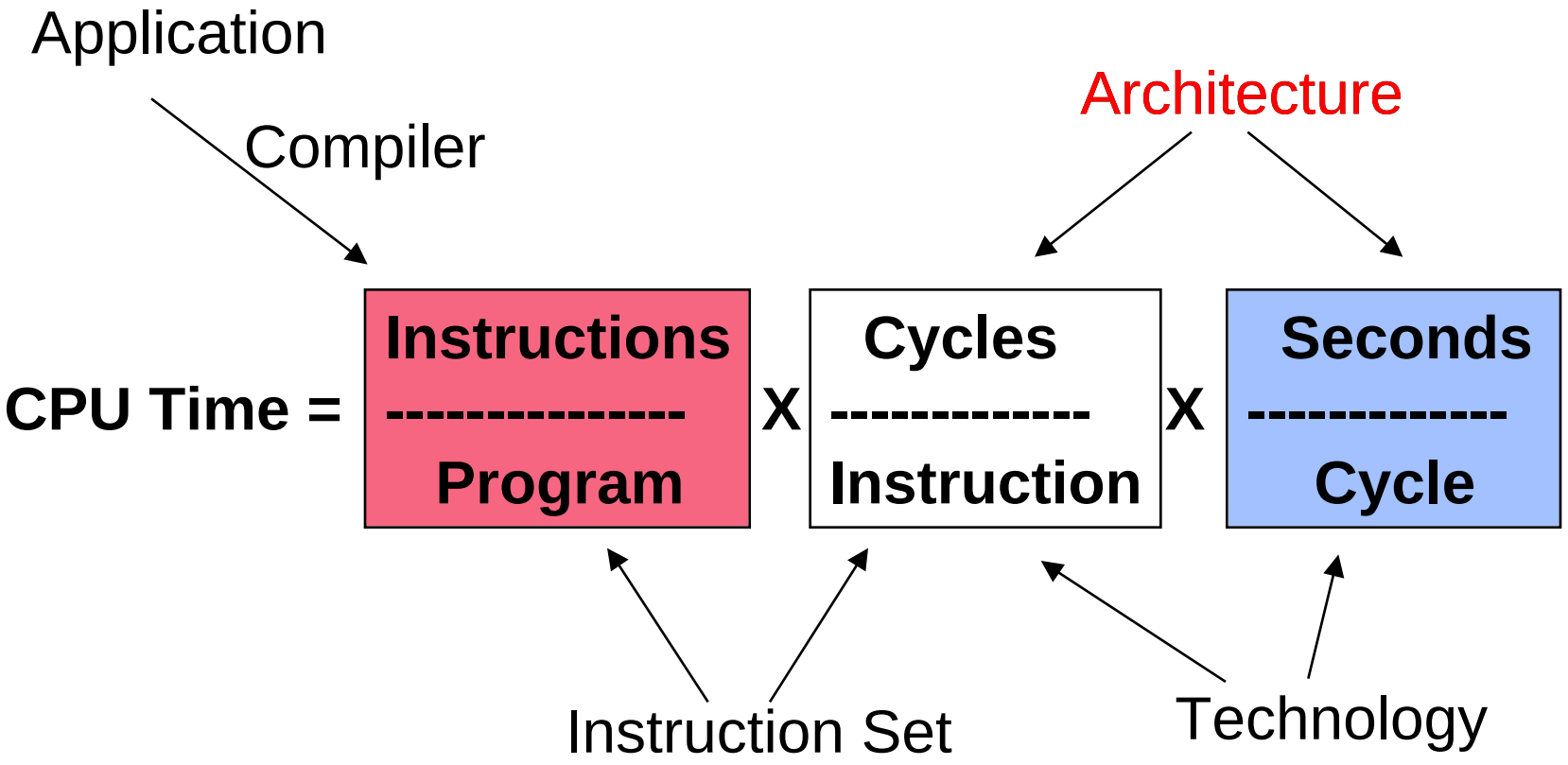
CPI as a Performance Metric

- "Average Cycles per Instruction"
- Dangerous for cross-platform comparison but useful for comparison with optimal values:
  - MIPS R10K (ASCI BlueMountain): 0.25
  - Itanium (watch this space for further news): 0.16
- "CPI Profiling"

$$\text{CPU time} = \text{CycleTime} * \sum_{i=1}^{n} \text{CPI}_i * \text{I}_i$$

- Look primarily at $\text{CPI}_{\text{stall}}$, $\text{CPI}_{\text{compute}}$, CPI_{mem}
  - Different methods for obtaining $\text{CPI}_{\text{stall}}$: nowadays measure it directly with HW counters

Aspects of Serial Performance



CPU Time = N_{inst} * CPI * Clock rate

Important Aspects of CPU Architecture

- Modern CPUs are
  - "Superscalar:" issue/execute >1 operation per clock period (CP) in separate functional units, typically 2-4 FLOPS, 2-4 mem, several ints, branches
    - Example: Itanium is a 6-issue machine
  - Pipelined: take 5-10 CP to complete but typically produce a result per CP after startup
    - Example: Itanium FP latency is 5 CP, IBM Pwr4 is 6 CP (IBM Pwr2 was 2 CP)
- Performance heavily dependent on compiler (and user) optimization

Instruction Level Parallelism

- Floating-point: 1 or more fused multiple/add ops.
  - IBM Power, Itanium: 2 FP units resulting in 4 FLOPS per CP (IBM P4 6-CP latency)
  - SGI R10K, DEC EV: 1 FP unit resulting in 2 FLOPS per CP
- Memory: multiple concurrent load/store pipelines
  - IBM Power: 4 mem access ops per CP
  - SGI R10K, DEC EV, Itanium: 2 mem access ops per CP
- *A fundamental aspect of RISC architecture is that the functional units can run independently. Therefore, FMAs can run in parallel with load/stores and other functions.* - **IBM Power4 Redbook**

Instruction Level Parallelism

```
      subroutine sub_1(n,x1,x2,x3,x4)
c gives peak speed for instance on IBM POWER2, SGI O2K; lat=2
C gives 2/3 peak speed on EV6; lat=4
      implicit real*8 (a-h,o-z)
      do 10,i=1,n
         y1=x1 + .500007500110d-05*x2 + .100001500023d-04*x3
         y2=x2 + .500007500110d-05*x1 + .100001500023d-04*x4
         y3=x3 + .100001500023d-04*x1 + .500007500110d-05*x4
         y4=x4 + .100001500023d-04*x2 + .500007500110d-05*x3

         x1=y1 + .500007500110d-05*y2 + .100001500023d-04*y3
         x2=y2 + .500007500110d-05*y1 + .100001500023d-04*y4
         x3=y3 + .100001500023d-04*y1 + .500007500110d-05*y4
         x4=y4 + .100001500023d-04*y2 + .500007500110d-05*y3
10    continue
      return
      end
```

Peak Performance

```
/bin/tcsh (ttyp1)
 14     16384  0.02   444.4 0.472E+00
   2.4345006669034152E+59  2.4345006669034152E+59  2.4345006669034148E+59
   2.4345006669034148E+59
t01->f77 -O3 -o peak peak.f -mips4 -64
t01->peak
 CYCLES, SPEED (MFLOPS), TIME (sec)
  0         1******     0.0 0.444E+03
  1         2  0.03   322.2 0.651E+00
  2         4  0.03   364.0 0.576E+00
  3         8  0.02   416.4 0.504E+00
  4        16  0.02   437.4 0.479E+00
  5        32  0.02   441.3 0.475E+00
  6        64  0.02   448.2 0.468E+00
  7       128  0.02   450.3 0.466E+00
  8       256  0.02   447.8 0.468E+00
  9       512  0.02   446.0 0.470E+00
 10      1024  0.02   445.1 0.471E+00
 11      2048  0.02   444.7 0.472E+00
 12      4096  0.02   444.6 0.472E+00
 13      8192  0.02   444.4 0.472E+00
 14     16384  0.02   444.4 0.472E+00
   2.4345006669034152E+59  2.4345006669034152E+59  2.4345006669034148E+59
   2.4345006669034148E+59
t01->
```

Code Restructuring for On-Chip Parallelism: Original Code

```
      subroutine length1(n,aa,tt)
      implicit real*8 (a-h,o-z)
      dimension a(n)
      tt=0.d0

      do 100, j=1,n
           tt=tt+a(j)*a(j)
  100   continue
      return
      end
```

Procedure: Unroll the loop to increase the number of independent computations in each iteration and keep the pipelines full.

Modified Code for On-Chip Parallelism

```
        subroutine length4 (n,a,tt)
c works correctly only if n is multiple of 4
        implicit real*8 (a-h,o-z)
        dimension a(n)
        t1=0.d0
        t2=0.d0
        t3=0.d0
        t4=0.d0
        do 100, j=1,n-3,4
c first floating point instruction unit, all even cycles
                  t1=t1+a(j+0)*a(j+0)
c first floating point instruction unit, all odd cycles
                  t2=t2+a(j+1)*a(j+1)
c second floating point instruction unit, all even cycles
                  t3=t3+a(j+2)*a(j+2)
c second floating point instruction unit, all odd cycles
                  t4=t4+a(j+3)*a(j+3)
100      continue
        tt= t1+t2+t3+t4
        return
        end
```

Software Pipelining

```
c first FP unit, first cycle:
c do one MADD (with t1 and a0 available in registers) and load a1:
    t1=t1+a0* a0
    a1=a(j+1)
c first floating point unit, second cycle:
c do one MADD (with t2 and t1 available in registers) and load a0 for next  iteration:
    t2=t2+a1*a1
    a0=a(j+0+4)
c second FP unit, first cycle:
c do one MADD (with t3 and a2 available in registers) and load a3:
    t3=t3+a2*a2
    a3=a(j+2)
c second FP unit, second cycle:
c do one MADD (with t4 and a3 available in registers) and load a2 for next iteration:
    t4=t4+a3*a3
    a2=a(j+1+4)
```

Intel Itanium Compiler

- **SWP report for loop at line 7924 in VQTERM in file hydronot100.f**

  Modulo scheduling was successful, but rotating register allocation failed because there were not enough rotating registers. Loop distribution, if it is legal, may help.

- **SWP report for loop at line 7906 in VQTERM in file hydronot100.f**

  Modulo scheduling was successful, but there was no overlap across iterations => loop not pipelined

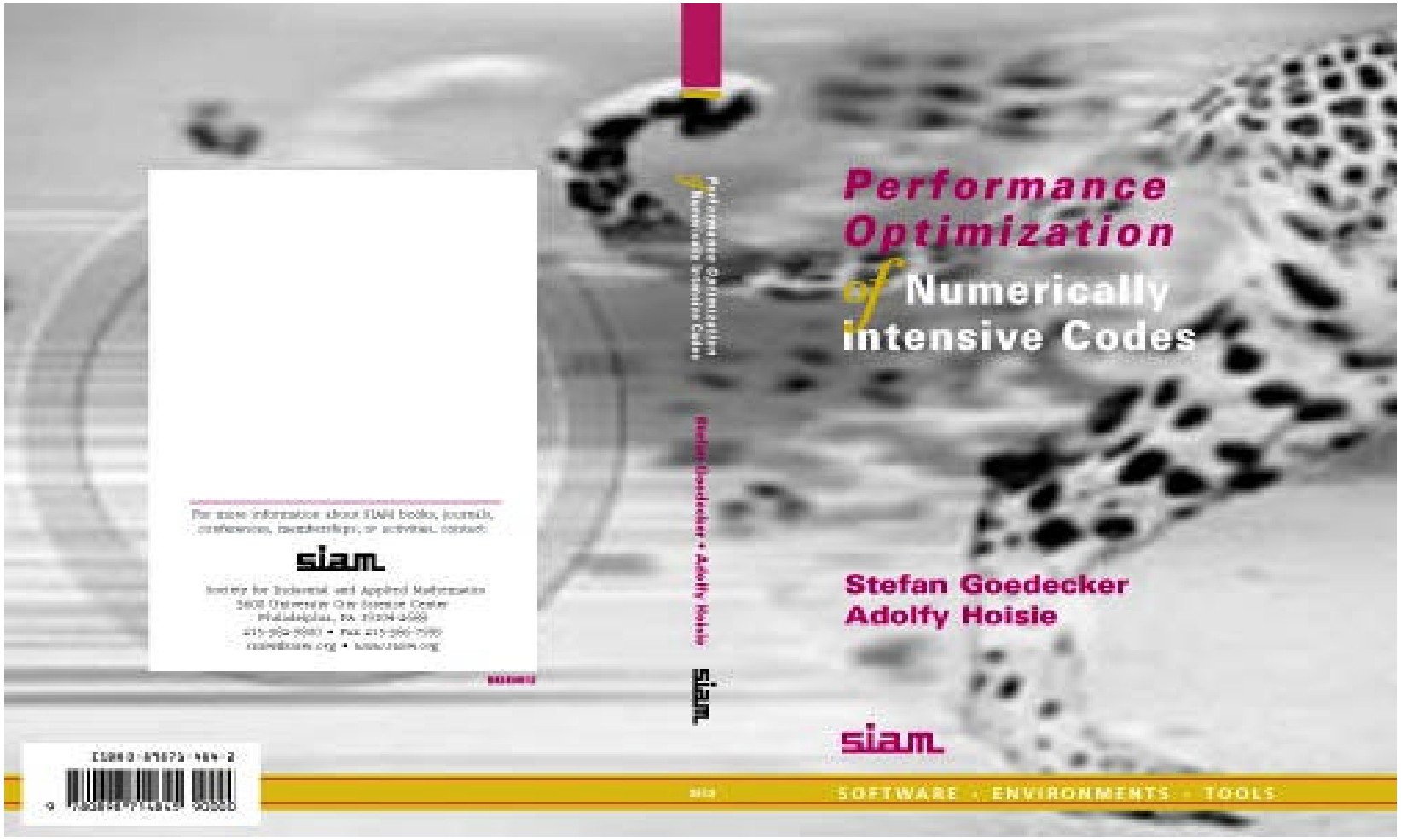
Summary of Optimization Techniques

- Restructure the code to minimize memory traffic and enable efficient software pipelining.

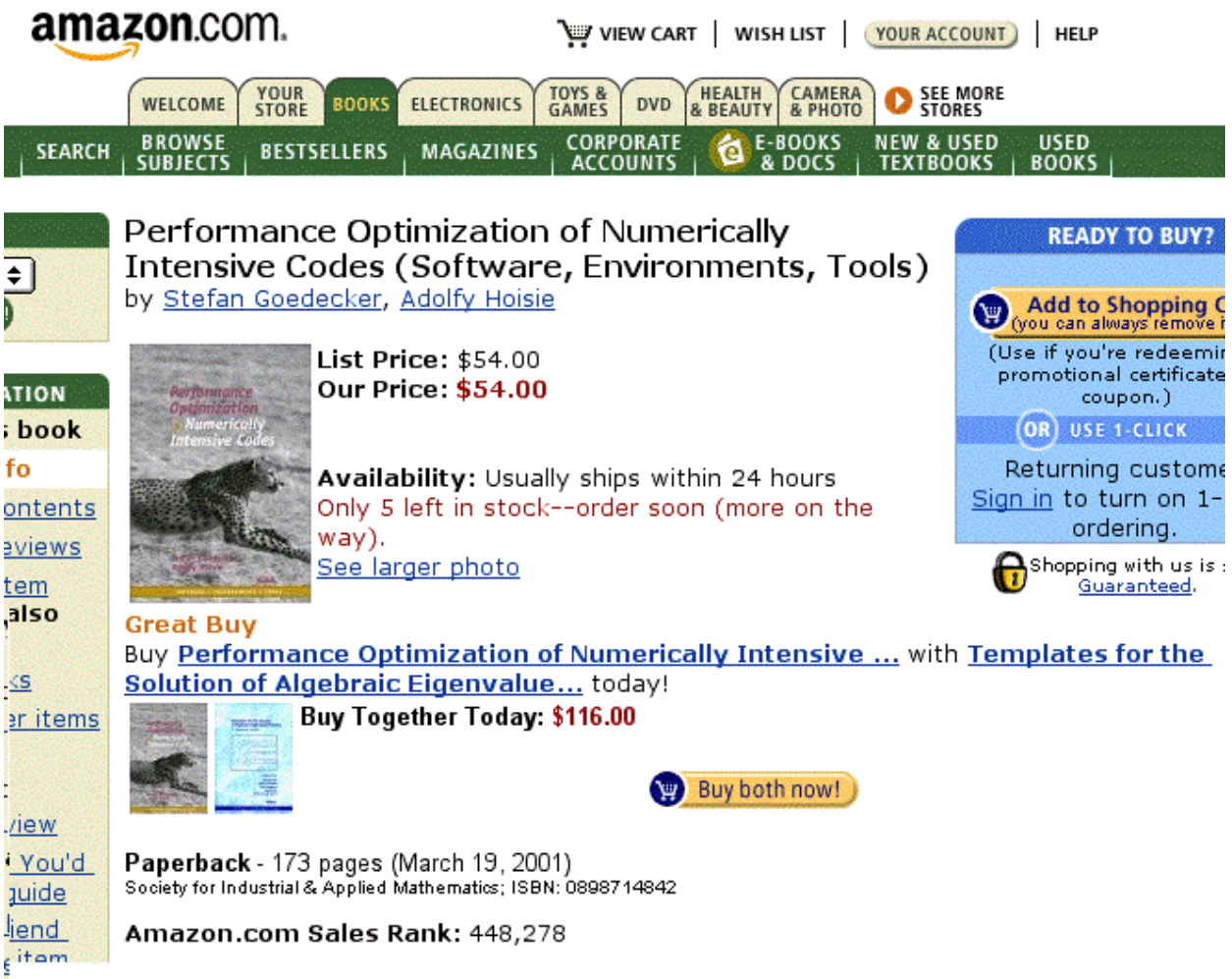
- Don’t rely on the compiler.
  - Compilers are getting better but they are not there yet: can do loop transformations only in simple cases, usually fail to produce optimal blocking;
  - they give up trying to SWP loops that vectorized easily 25 years ago.

- Read the book / take the full tutorial…

The Book


Performance Optimization of Numerically Intensive Codes
Stefan Goedecker
Adolfy Hoisie
siam
SOFTWARE • ENVIRONMENTS • TOOLS

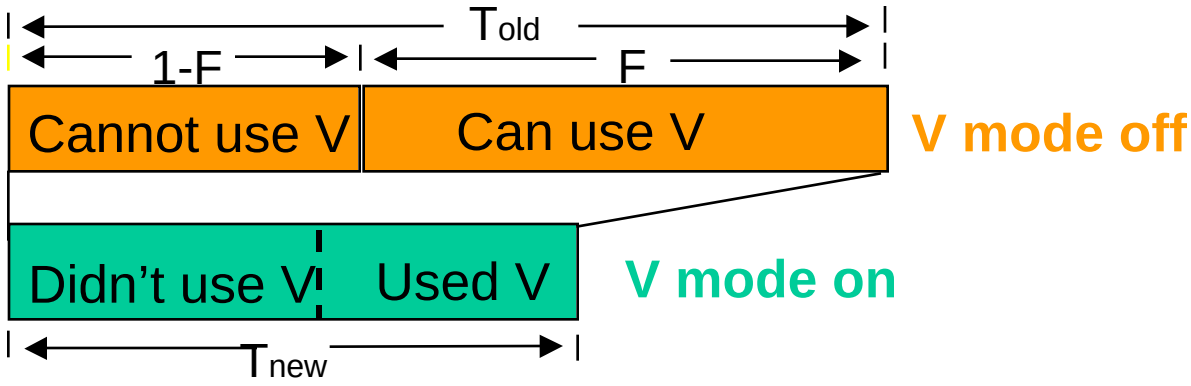
The Book



- http://www.amazon.com/exec/obidos/ASIN/0898714842/qid%3D1011019032/ref%3Dsr%5F11%5F0%5F1/002-2604300-3240818

Amdahl's Law

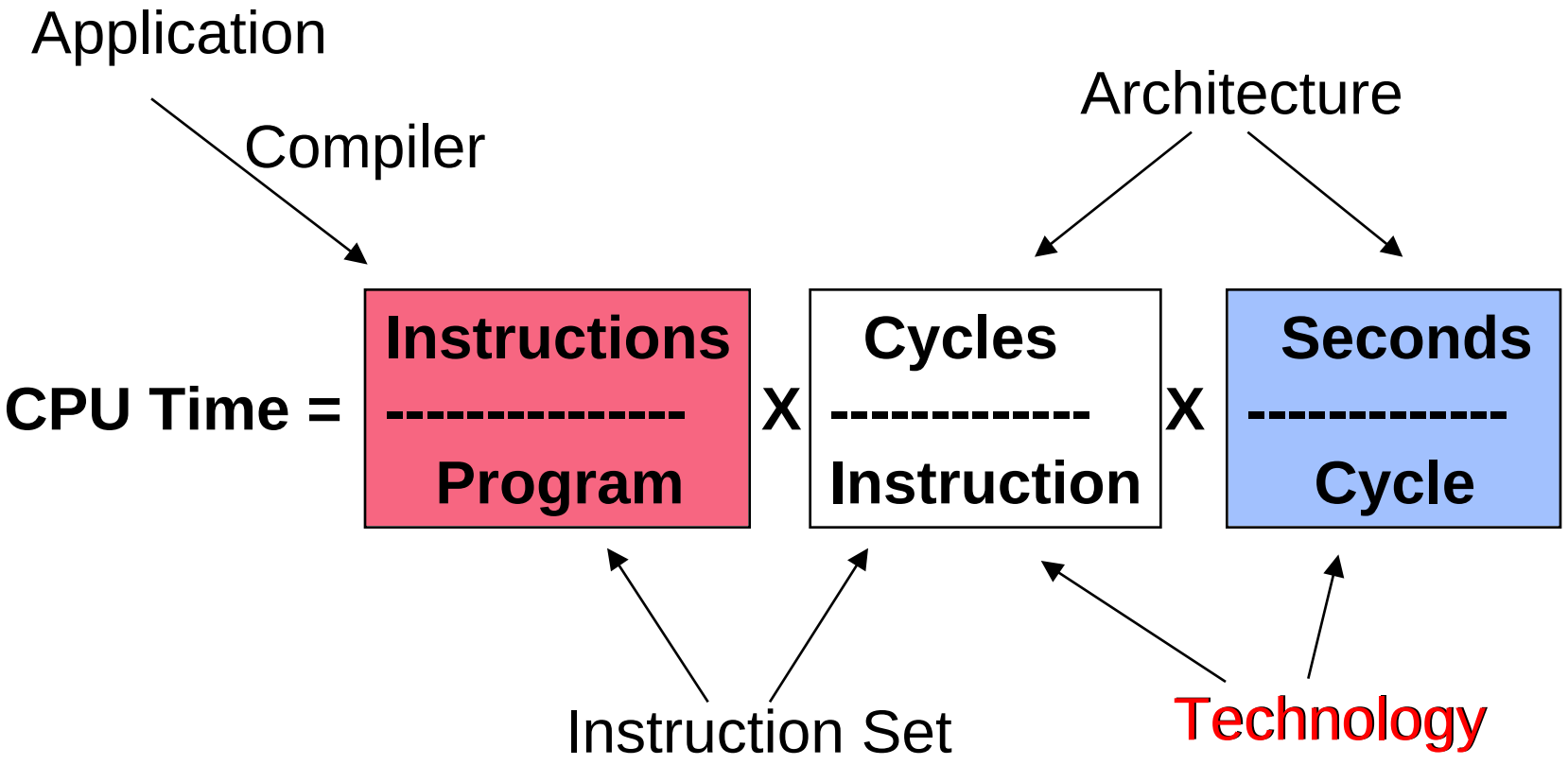
Suppose that enhancement V accelerates a fraction F of a task by a factor S, and the remainder of the task is unaffected.



$$\text{Speedup} = \frac{\text{Time}_{old}}{\text{Time}_{new}} = \frac{1}{(1 - \text{Fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}}}$$

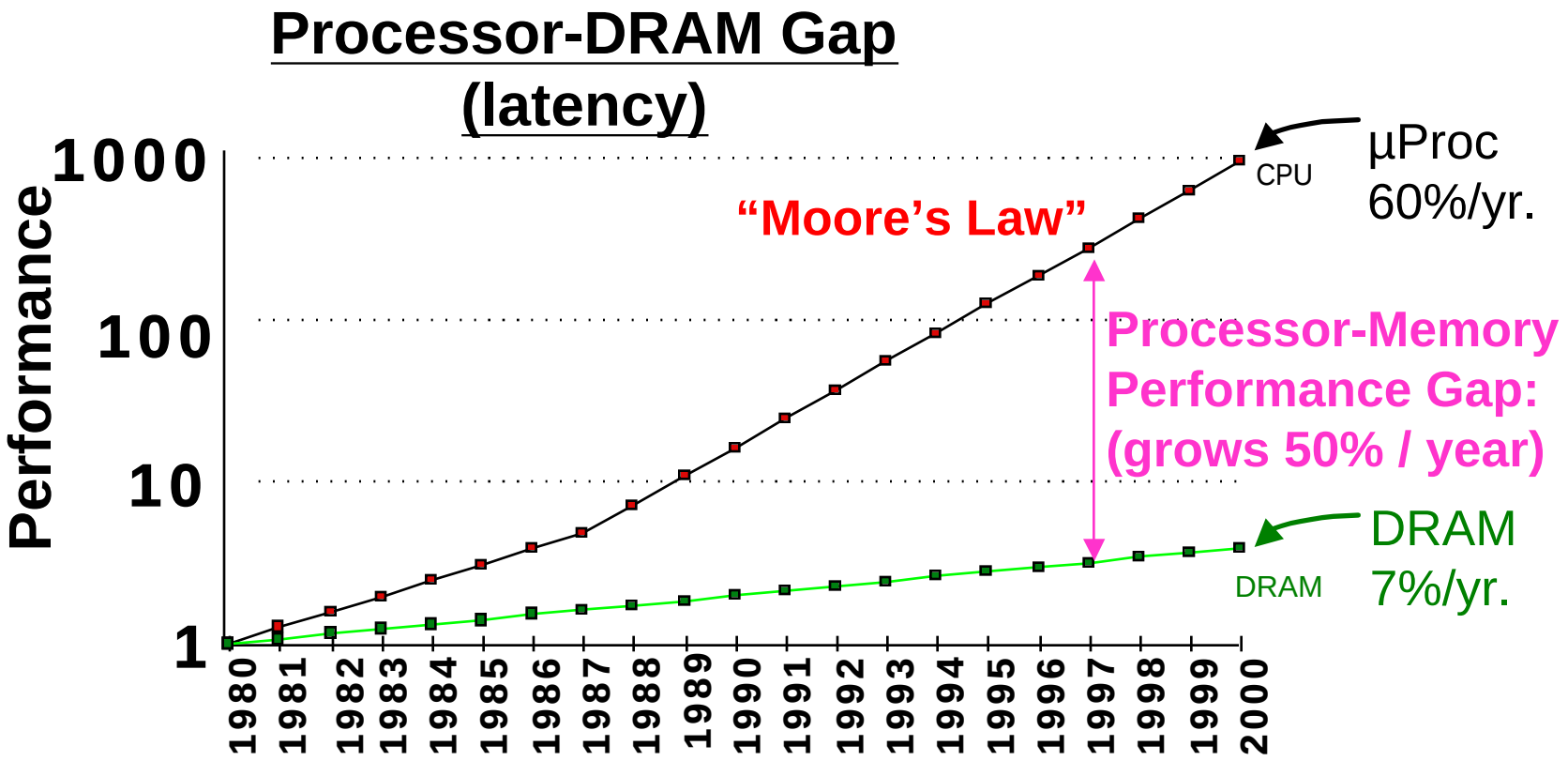
- Amdahl's law applies to any overhead, not just limited concurrency

Aspects of Serial Performance



CPU Time = N_{inst} * CPI * Clock rate

Memory Performance (Latency)



From D. Patterson, CS252, Spring 1998 ©UCB

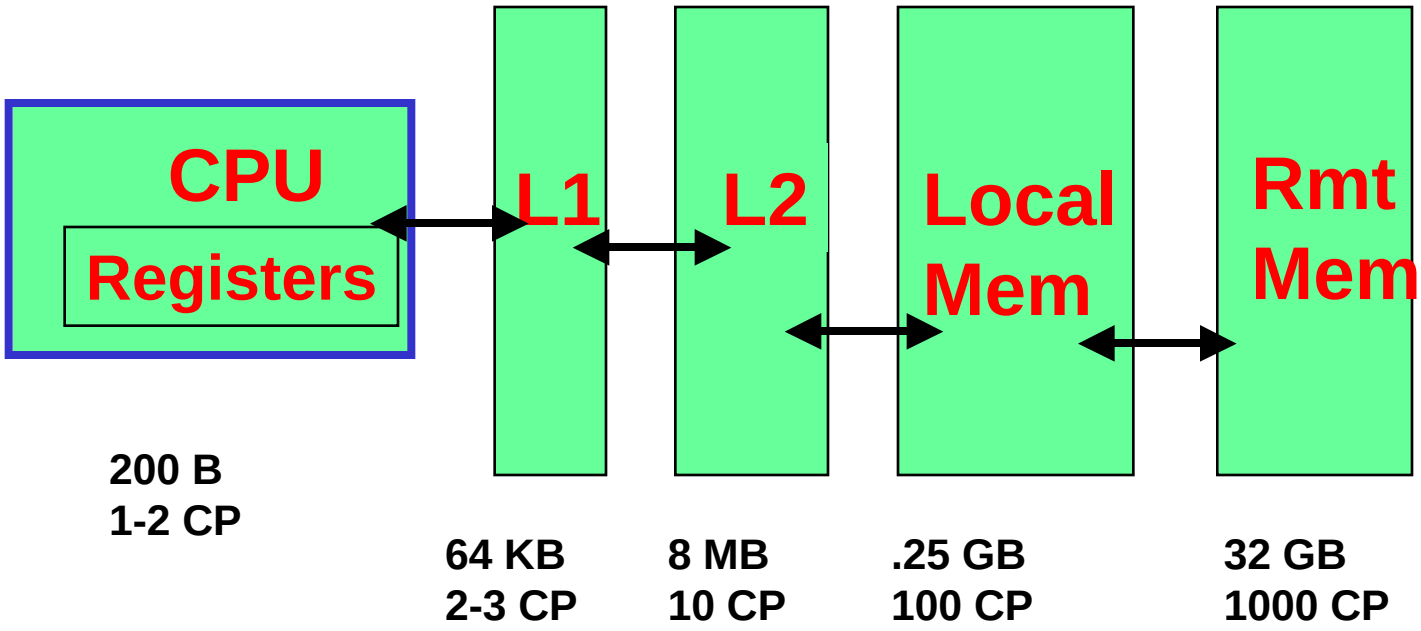
Memory Performance (Latency)

Machine	Latency (ns)	CPU CP (ns)	Ratio	Issue Rate	Number of Instructions
1st Alpha	340	5.0	68	2	136
2nd Alpha	266	3.3	80	4	320
3rd Alpha	180	1.0	180	4	720

1/2X latency x 5X CPU CP x 2X issue rate = ~5x

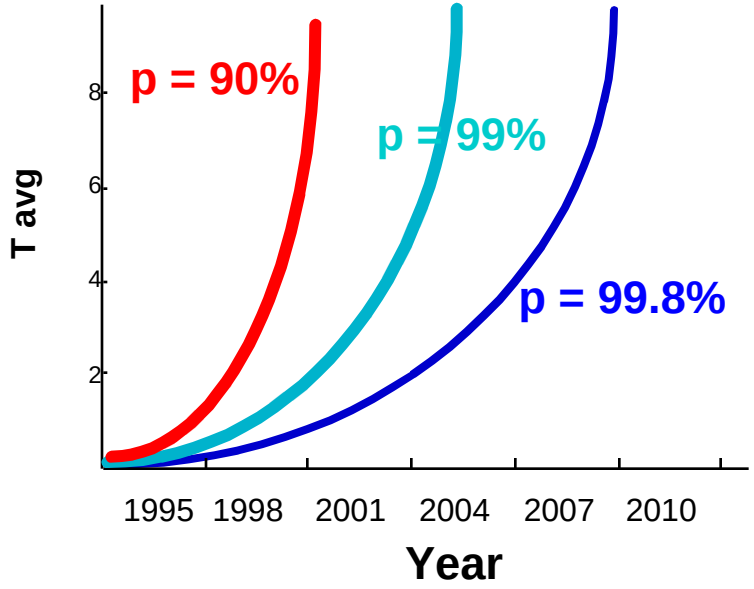
Memory Performance (Latency)

- New architectures include features that attempt to
  - **Hide** memory latency (cache, non-blocking loads, ILP)
  - **Reduce** memory latency (faster DRAM, faster interconnects)
  - Allow you to "measure" effectiveness of memory hierarchy



Amdahl's Law and the Memory Gap: Hitting the "Memory Wall"

- **Avg Time = p x Tcomp + (1-p) x Tmem**
- **S_{comp} increasing at ~ *60%* per year**
- **S_{mem} increasing at ~ *7%* per year**



Wulf, W.A & McKee, S. "Hitting the Memory Wall: Implications of the Obvious," Comp. Arch. News, March, 1995.

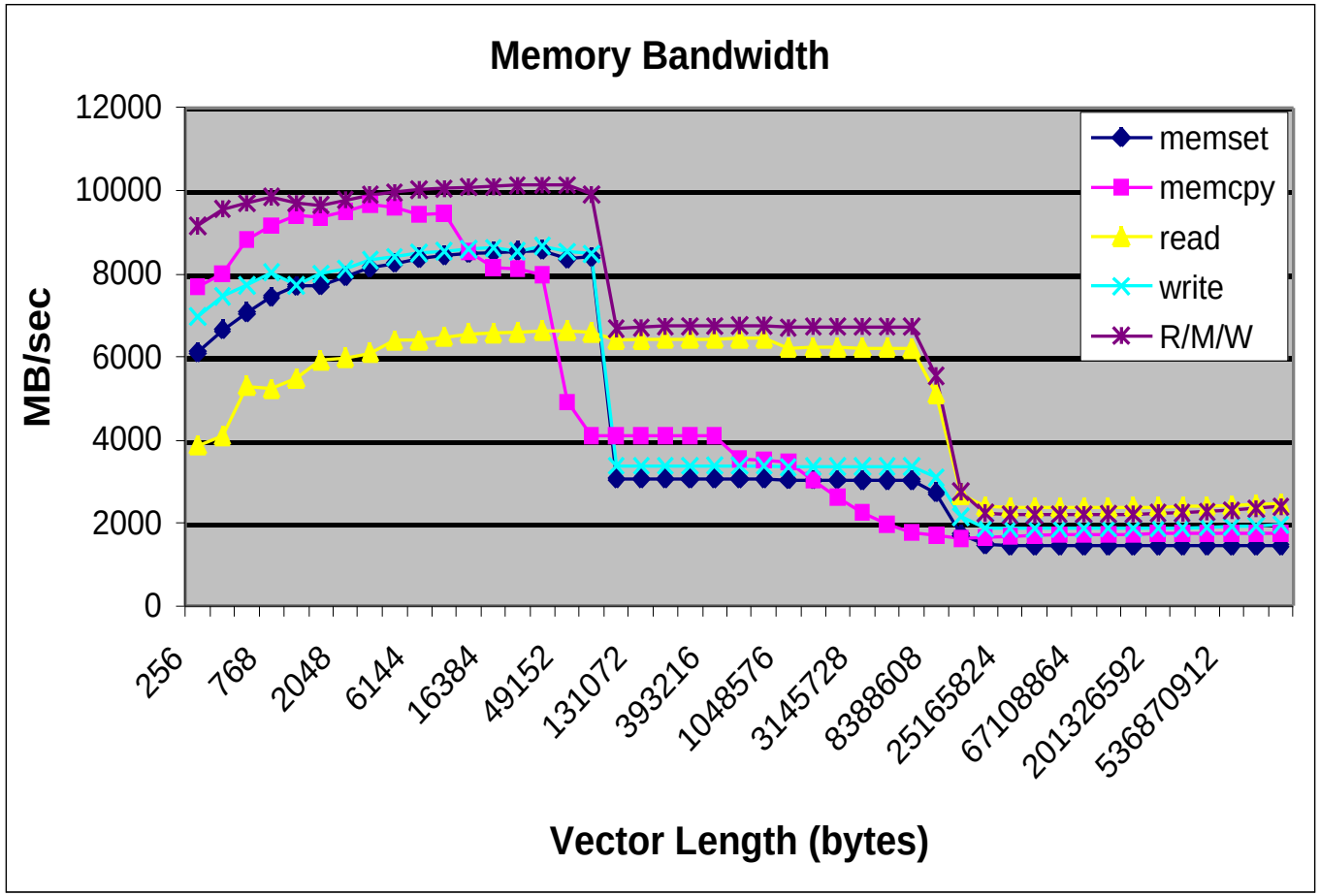
Cache Architectures

- SGI R10K (250 MHz)
  - 32-KB L1
  - 4-MB L2
  - Latencies:
    - **L1: ≤1 CP**
    - **L2: ≤11 CP**
- Intel Itanium (800 MHz)
  - 16-KB L1, 4-way, 32-B line
  - 96-KB L2, 6-way, 64-B line
  - 2-MB L3, 4-way, 64-B line
  - Latencies:
    - **L1: ≤ 2 CP**
    - **L2: ≤ 12 CP**
    - **L3: ≤ 20CP**
- IBM Power4 (1300 MHz)
  - 64-KB L1, 2-way, 128-B line
  - 1.4-MB L2 (shared)
  - 128-MB L3 8-way (shared)
  - Latencies:
    - **L1: ≤ 4 CP**
    - **L2: ≤ 14 CP**
    - **L3: ≤ ~300 CP**
- DEC EV68 (1000 MHz) (Compaq ES45)
  - 64-KB L1, 2-way
  - 8-MB L2
  - Latencies:
    - **L1: ≤3 CP**
    - **L2: ≤20 CP**

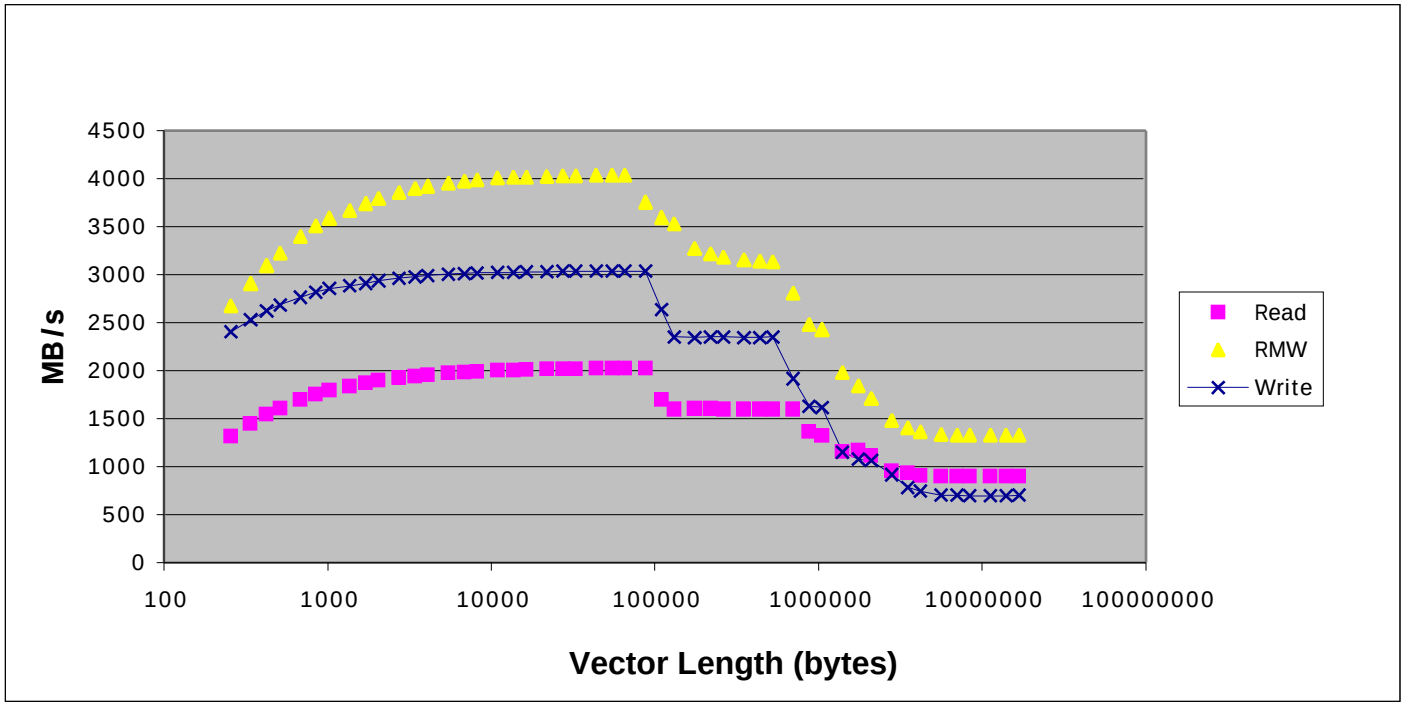
Caches

- Lots of technology, lots of transistors (50-75% of the chip)
- How well are they working?

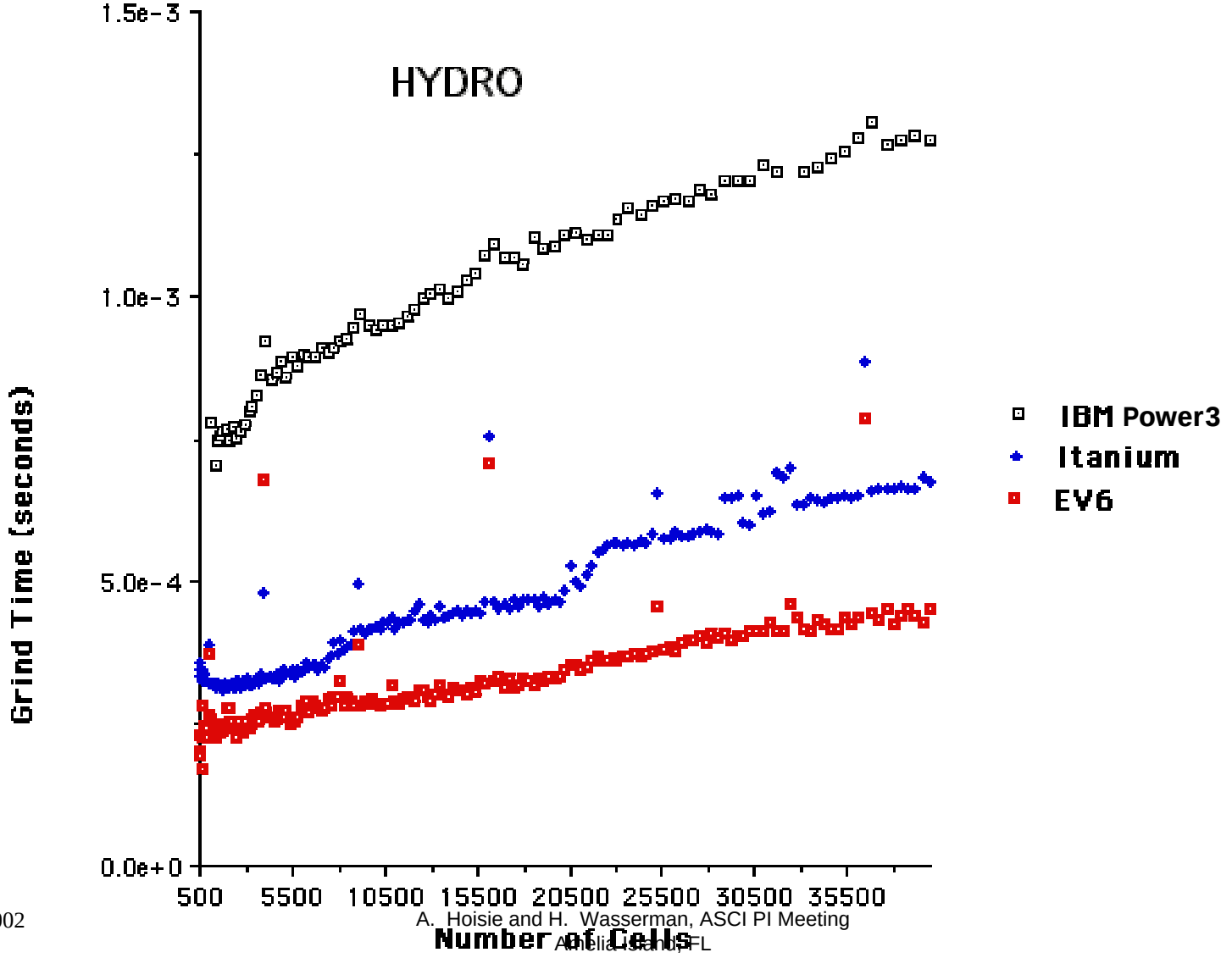
Measured Memory BW on DEC EV6



Measured Memory BW on ITANIUM



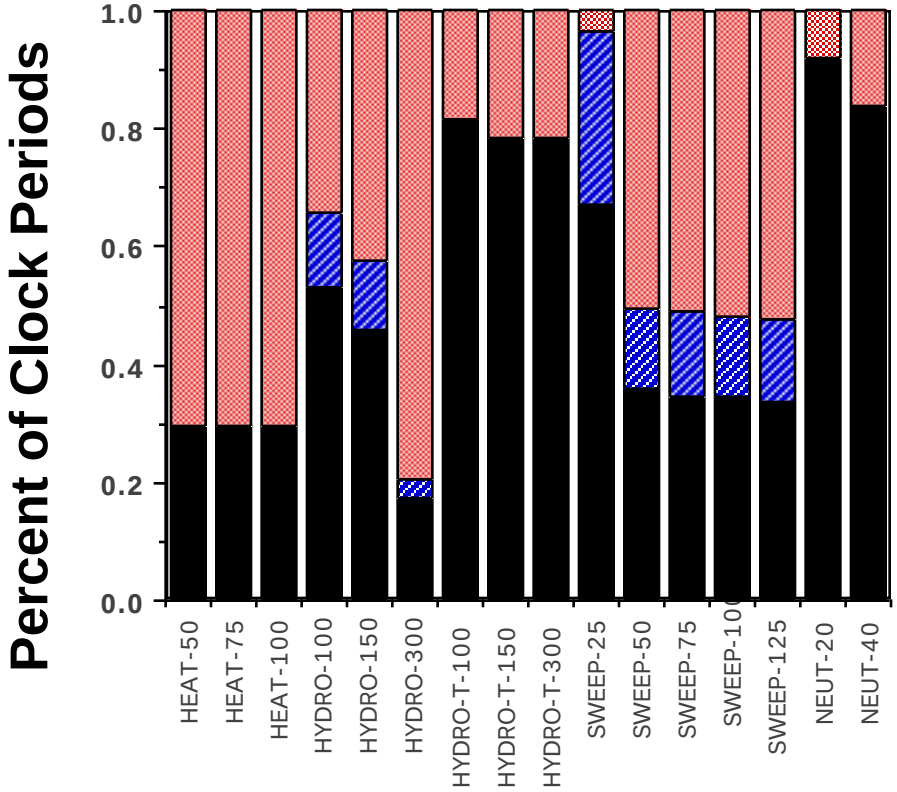
Code Performance as a Function of Size



How Well Do Caches Work?

- So far we looked at performance in terms of observed mem-to-CPU BW and/or CPU time.
- We can also see the effect of memory performance in CPI…

Performance Evaluation via CPI Profiling



% of CPI spent idle waiting on memory

% of CPI spent idle waiting on L2 cache

% of CPI doing computation (not idle)

	T2 O2K	Tm O2K	CPIo O2K	CPI O2K
HEAT	0	60	.74	1.7
HYDRO	2	50	.89	1.7
HYDRO-T	0	11	.9	0.95
SWEEP	11	43	.88	1.5
NEUT	2.2	205	.77	0.80
(NOMINAL)	11	205	.25	.25

O. Lubeck, Y. Luo, H. Wasserman and F. Bassetti **"Performance "Evaluation of the SGI Origin2000: A Memory-Centric Characterization of LANL ASCI Applications," *Proc. SC97***

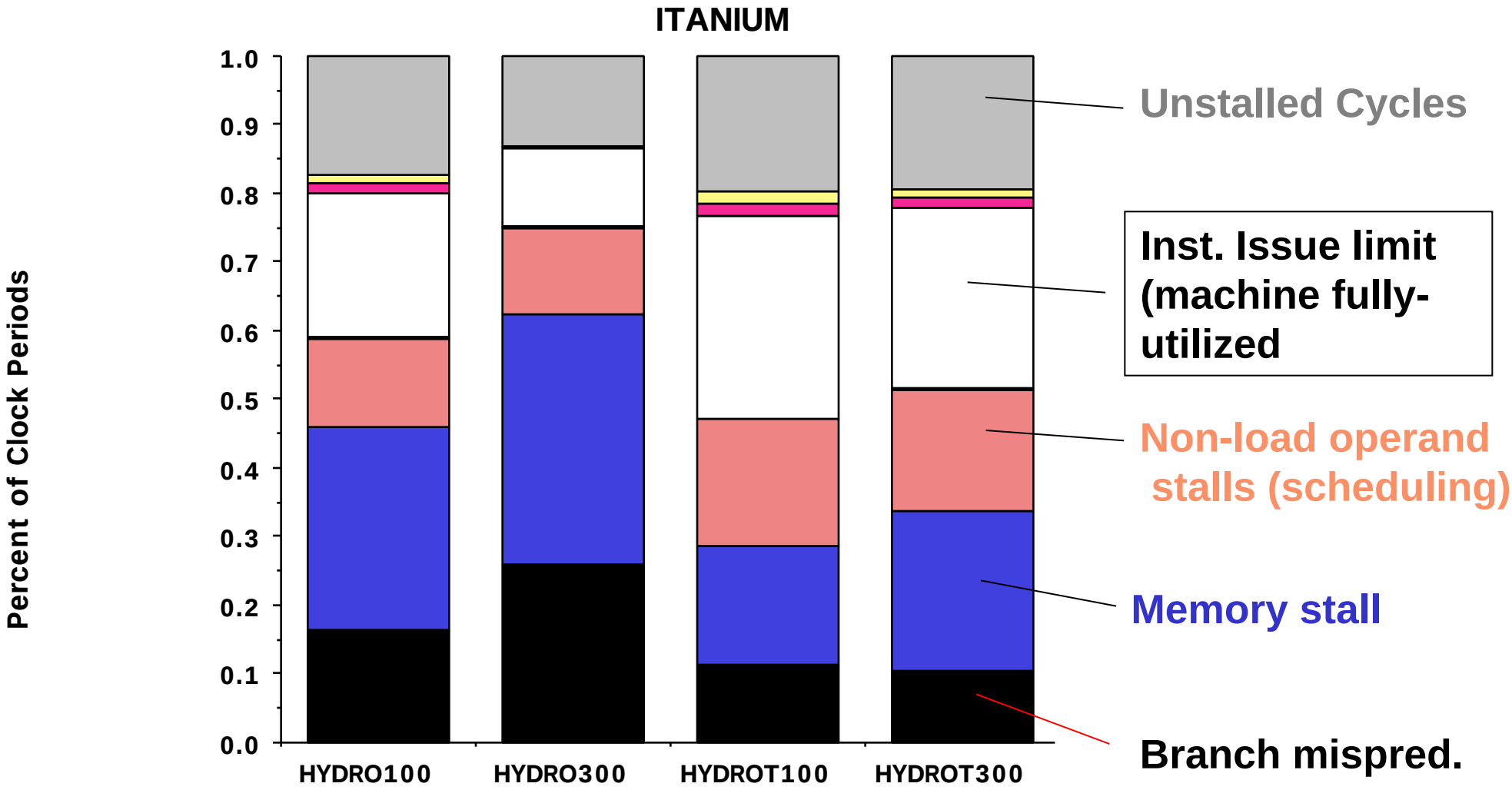
Putting It All Together

- CPU Time = N_{inst} * CPI * Clock rate
- Where Have All the Cycles Gone?…

Study of HYDRO on Itanium

- Two versions of this 2-D code:
  - HYDRO uses stride-***n*** access, where ***n*** is the (1-D) size of the grid
  - HYDRO-T has been transposed so that most accesses are now stride-1.
- Two problem sizes: 100 x 100 fits in L2, 300 x 300 does not.

Where Have All the Cycles Gone?



Study of HYDRO on Itanium: Observations

- Increase in memory access CPI is apparent in HYDRO300
  - But reason for increase in branch-mispredict CPI is unclear; more investigation needed
- Transposed version shows very little increase in memory access CPI as a function of problem size.
- Not shown: Actual CPI values:
  - HYDRO100 = 0.56 HYDROT100 = 0.54
  - HYDRO300 = 1.09 HYDROT300 = 0.56
  - These values are still large relative to optimal value
  - You can see both technology (memory) and architecture (SWP) effects in the chart - the transposed version still needs further optimization to improve SWP.

Los Alamos
NATIONAL LABORATORY

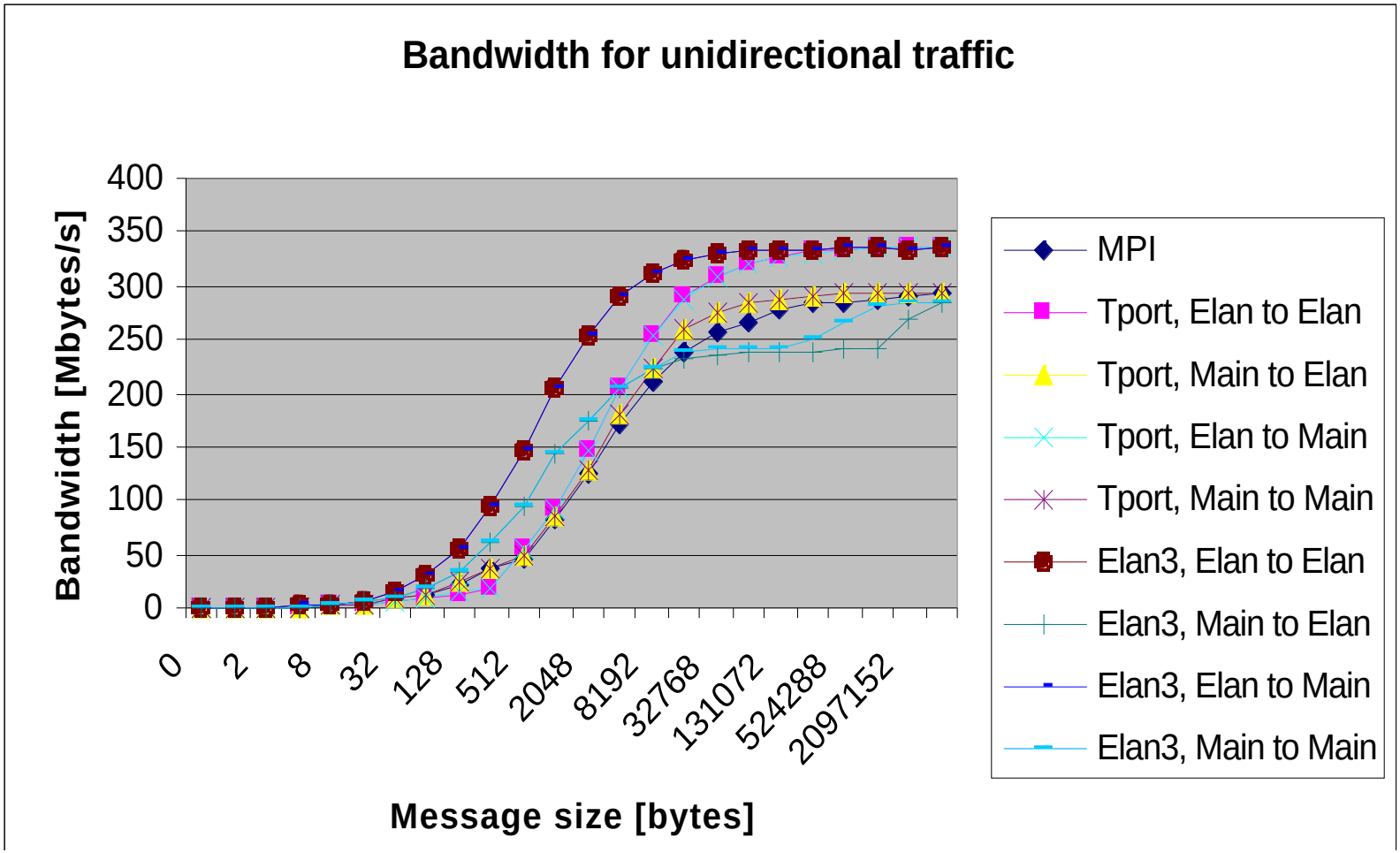
Multiprocessor Performance Modeling

ASCI PI January 8, 2002

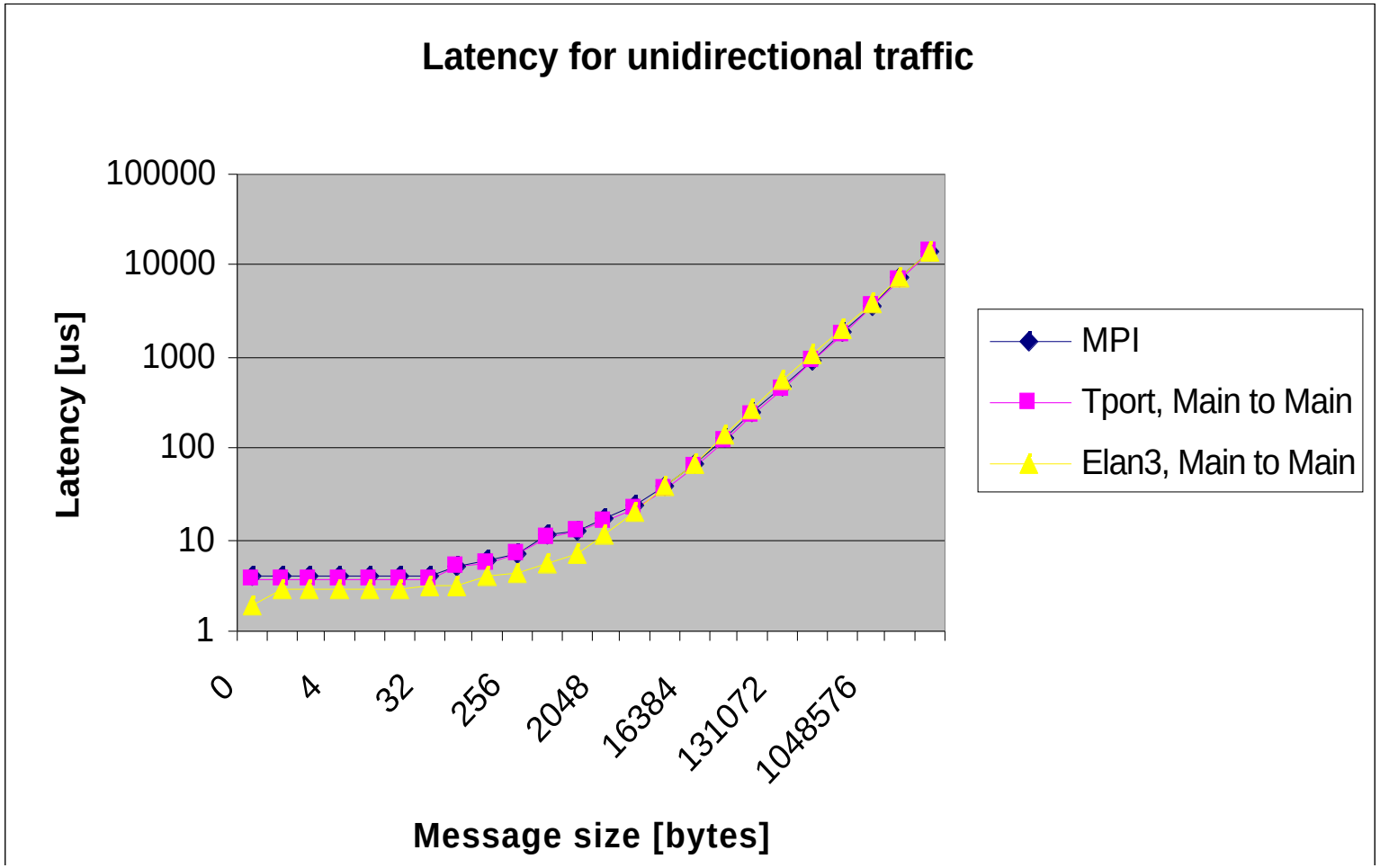
Compaq ES45

- ES45s with memories ranging from 8-16 Gbytes
- 4 processors/box clocked at 1 GHz
- 2 rails Quadrics interconnect
- Tru64
- Following slides are some basic communication kernel performance measurements from this system.

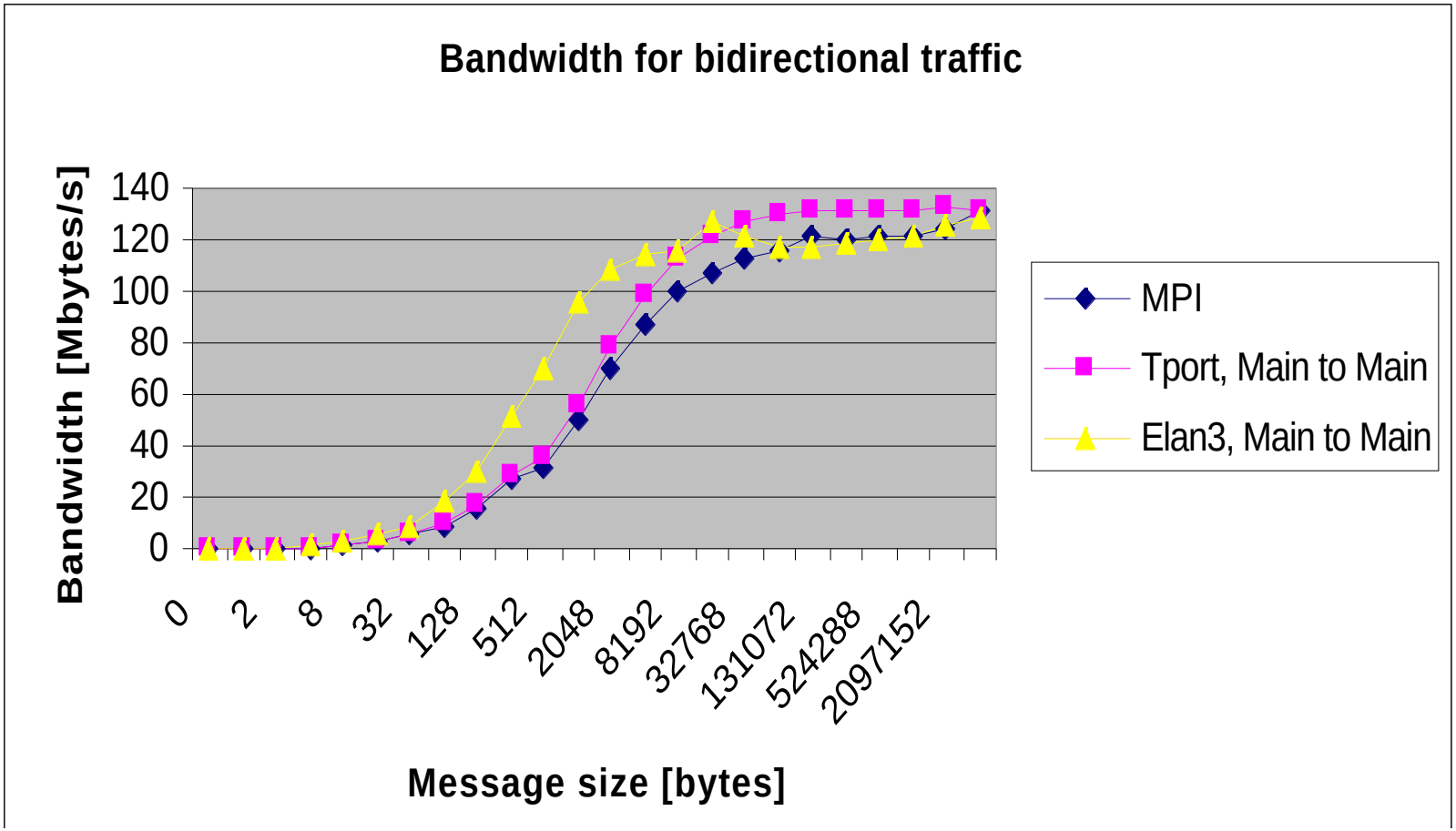
Compaq ES45


Bandwidth for unidirectional traffic
Bandwidth [Mbytes/s]
400
350
300
250
200
150
100
50
0
0
2
8
32
128
512
2048
8192
32768
131072
524288
2097152
Message size [bytes]
MPI
Tport, Elan to Elan
Tport, Main to Elan
Tport, Elan to Main
Tport, Main to Main
Elan3, Elan to Elan
Elan3, Main to Elan
Elan3, Elan to Main
Elan3, Main to Main

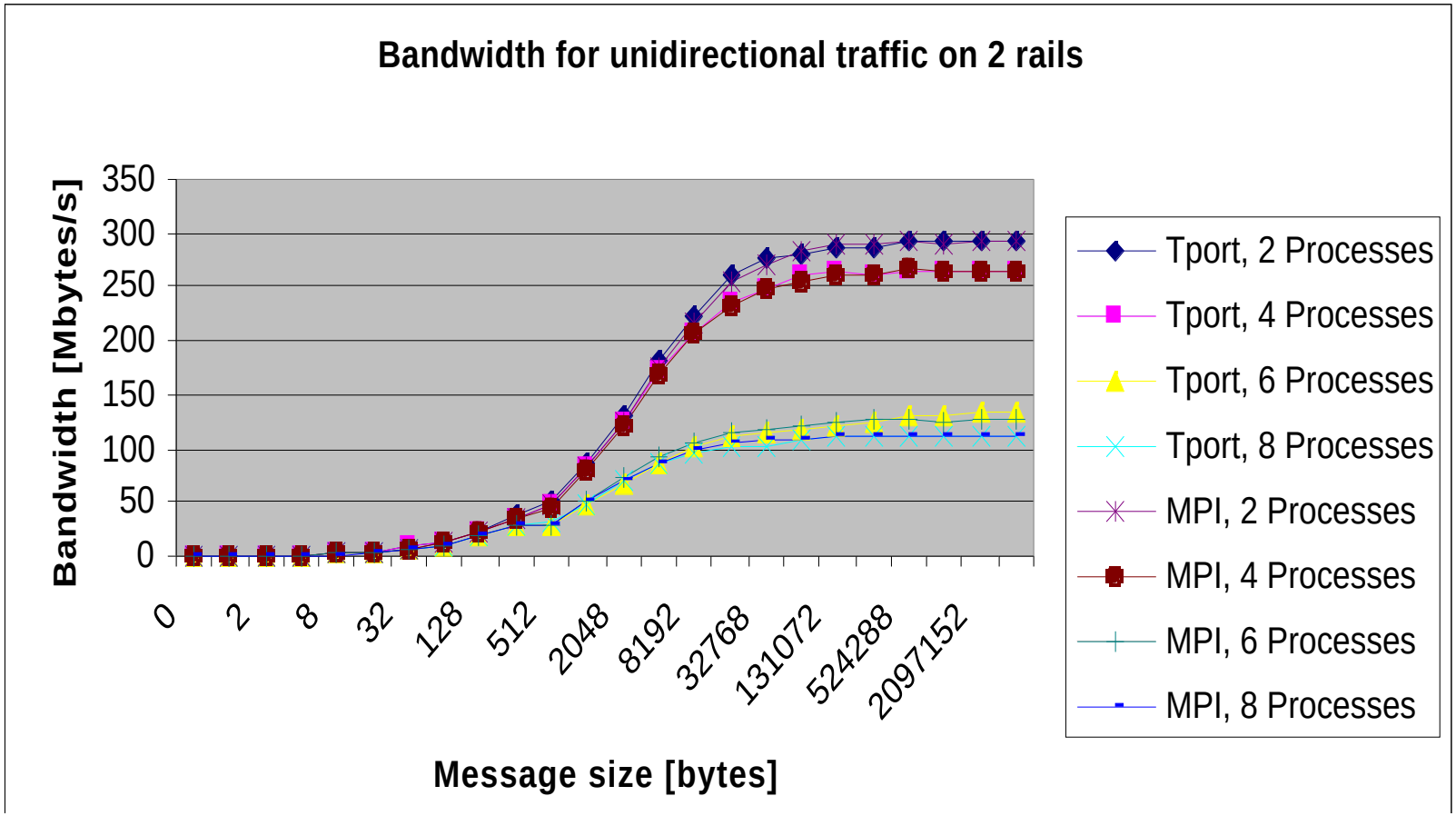
Compaq ES45


Latency for unidirectional traffic
Latency [us]
100000
10000
1000
100
10
1
0
4
32
256
2048
16384
131072
1048576
Message size [bytes]
MPI
Tport, Main to Main
Elan3, Main to Main

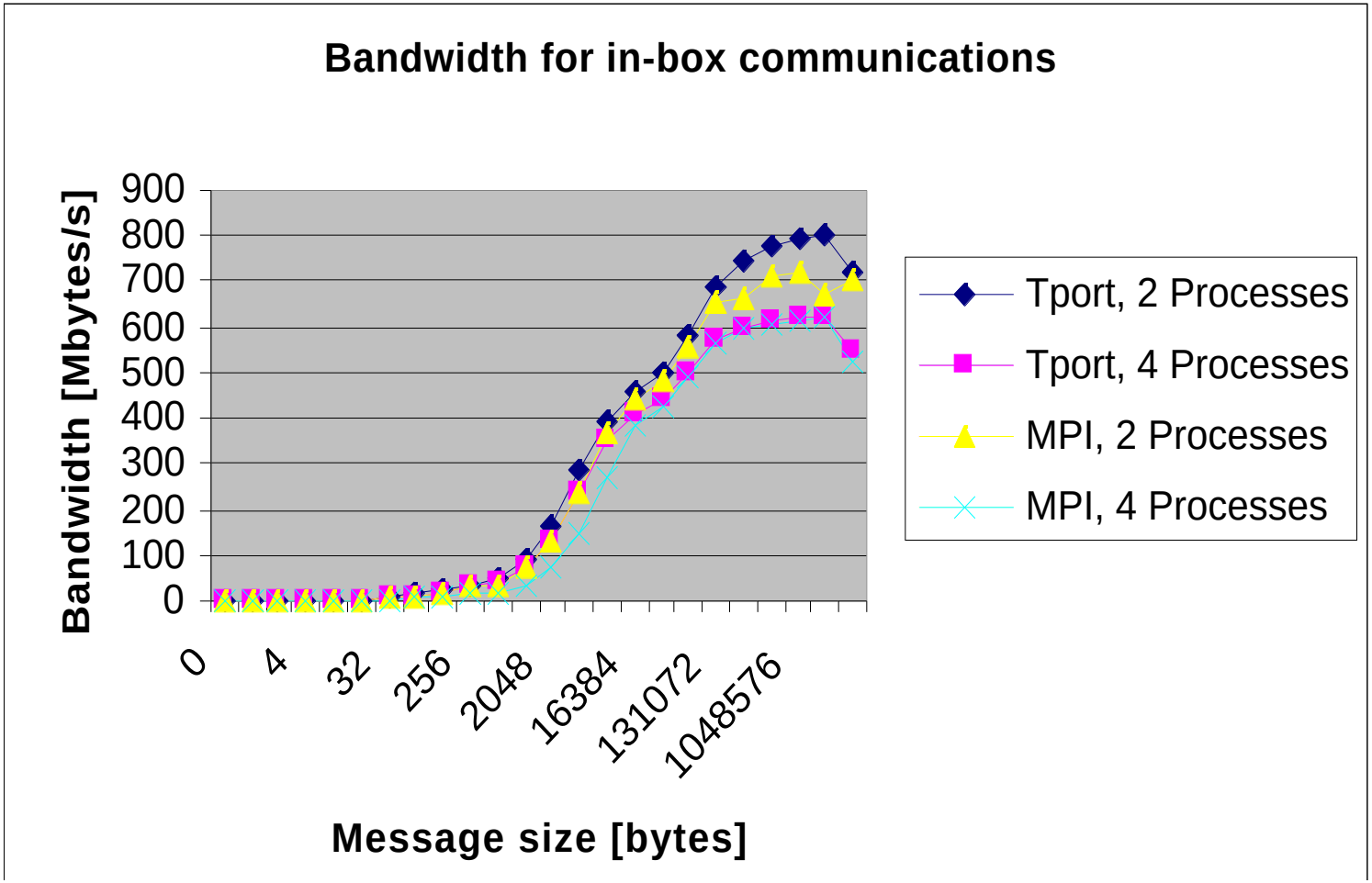
Compaq ES45


Bandwidth for bidirectional traffic
Bandwidth [Mbytes/s]
140
120
100
80
60
40
20
0
0
2
8
32
128
512
2048
8192
32768
131072
524288
2097152
Message size [bytes]
MPI
Tport, Main to Main
Elan3, Main to Main

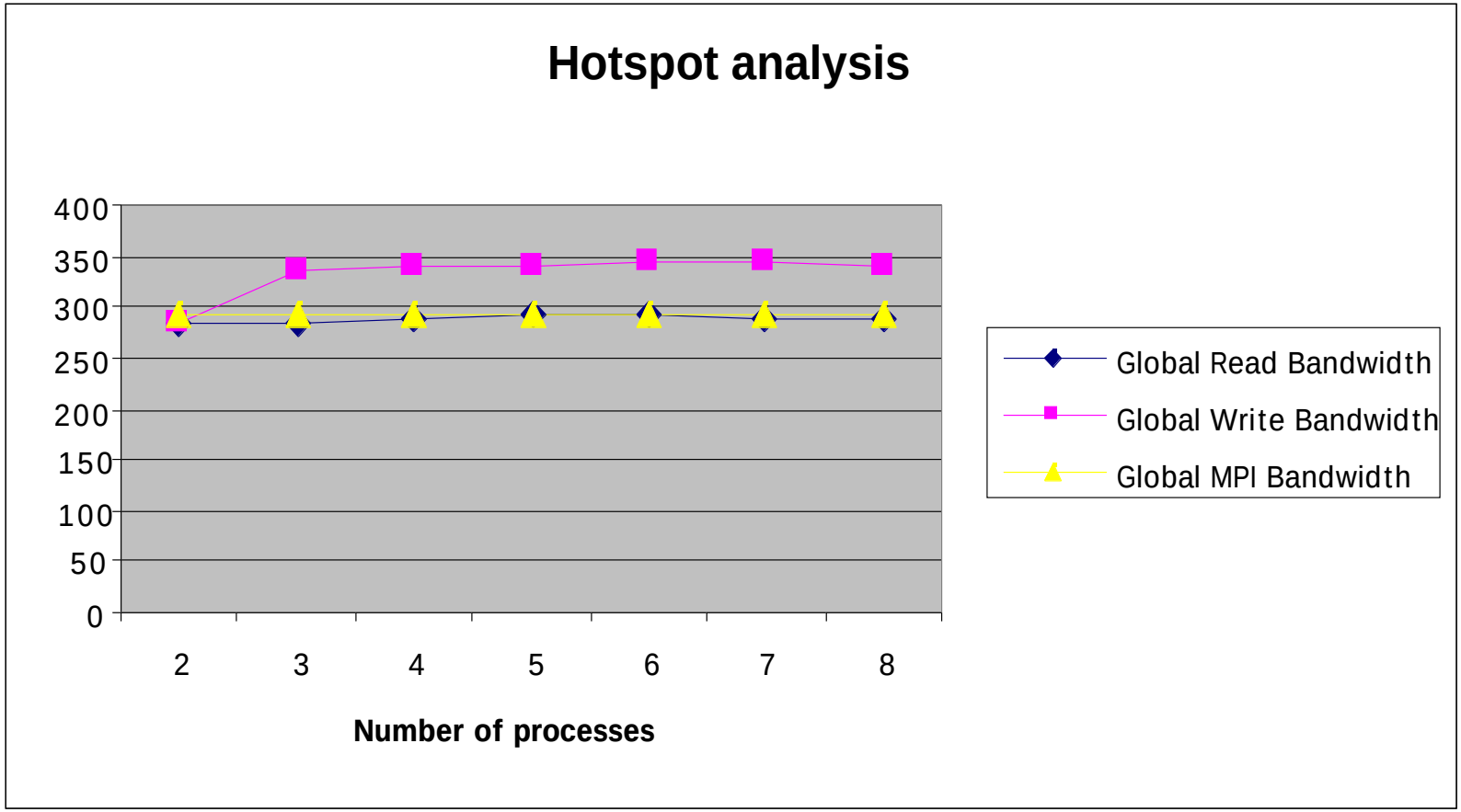
Compaq ES45


Bandwidth for unidirectional traffic on 2 rails
Bandwidth [Mbytes/s]
350
300
250
200
150
100
50
0
0
2
8
32
128
512
2048
8192
32768
131072
524288
2097152
Message size [bytes]
Tport, 2 Processes
Tport, 4 Processes
Tport, 6 Processes
Tport, 8 Processes
MPI, 2 Processes
MPI, 4 Processes
MPI, 6 Processes
MPI, 8 Processes

Compaq ES45


Bandwidth for in-box communications
Bandwidth [Mbytes/s]
900
800
700
600
500
400
300
200
100
0
0
4
32
256
2048
16384
131072
1048576
Message size [bytes]
Tport, 2 Processes
Tport, 4 Processes
MPI, 2 Processes
MPI, 4 Processes

Compaq ES45


Hotspot analysis
400
350
300
250
200
150
100
50
0
2
3
4
5
6
7
8
Number of processes
Global Read Bandwidth
Global Write Bandwidth
Global MPI Bandwidth

Performance analysis

- Understand current performance of architectures & applications
- Measuring performance is of limited use:
  - current implementation of code
  - currently available architectures
  - impossible to distinguish between real performance and machine idiosyncrasies
- Design space is Multidimensional
  - runtime = **f(** microprocessor performance, memory hierarchy, network characteristics, compiler/language etc. **)**
- Performance Characterization of the ASCI workload and ASCI Architectures
  - in-depth performance analysis and modeling done for important ASCI workload

…cont'd…performance is a multidimensional space

- **problem size**
- **# of processors**
- **architectural design (size, topology, etc)**
- **communication parameters**
- **computation parameters**
- **optimal (problem) blocking sizes**
- **target optimization (e.g., runtime, problem size)**

Performance Analysis Methods

- Analytical (algorithmic analysis)
- Statistical
  - System workload performance
  - Mean Value Analysis
- Queuing theory
- Experimental
  - Simulation
  - Benchmarking
  - Trace-driven experiments

Selection of Performance Analysis Method...

" More than any other time in history, mankind faces a cross-roads. One path leads to despair and utter hopelessness. The other, to total extinction. Let us pray we have the wisdom to choose correctly "

- ***Woody Allen***

The "Fundamental" Equation of Modeling

$$T_{run} = T_{computation} + T_{communication} - T_{overlap}$$

- **$T_{computation}$ is easiest to model. A coarse approximation is based on the number of grid points, characteristic Mflop rate, and a sensitivity analysis for cache behavior.**
- **$T_{communication}$ is trickier. It depends on the type of communication kernels (blocking, non-blocking), point-to-point or global communications, communication parameters, network topology, contention. The linear model (latency-bandwidth) or LogGP can be utilized.**
- **$T_{overlap}$ is the hardest. It depends on algorithmic overlap, communication /computation overlap in hardware, load balancing, contention, runtime variability, overall machine load.**

Performance Modeling

- Encapsulate performance characteristics
- Parameterized in terms of:
  - Problem sizes
  - Architecture (communication, CPU, memory)
  - Mapping (parallelization strategy etc)

Performance Engineering

- Performance-engineered system: The components
- (application and system) are ***parameterized*** and ***modeled***, and a ***constitutive model is proposed*** and ***validated***.

- ***Predictions*** are made based on the model. The model is meant to be ***updated, refined***, and ***further validated*** as new factors come into play.

Uses of Predictive Performance Models: A) Architecture and B) Application Design Exploration

- **No need for implementation**
- **Fast exploration (analytical based model)**
- **Use model to explore:**
  - **Change of existing Architectures**
    - **e.g. change in sub-system performance (increased communication bandwidth, upgrade of CPU etc)**
  - **Future Architectures (non-existing)**
    - **Compare alternatives**

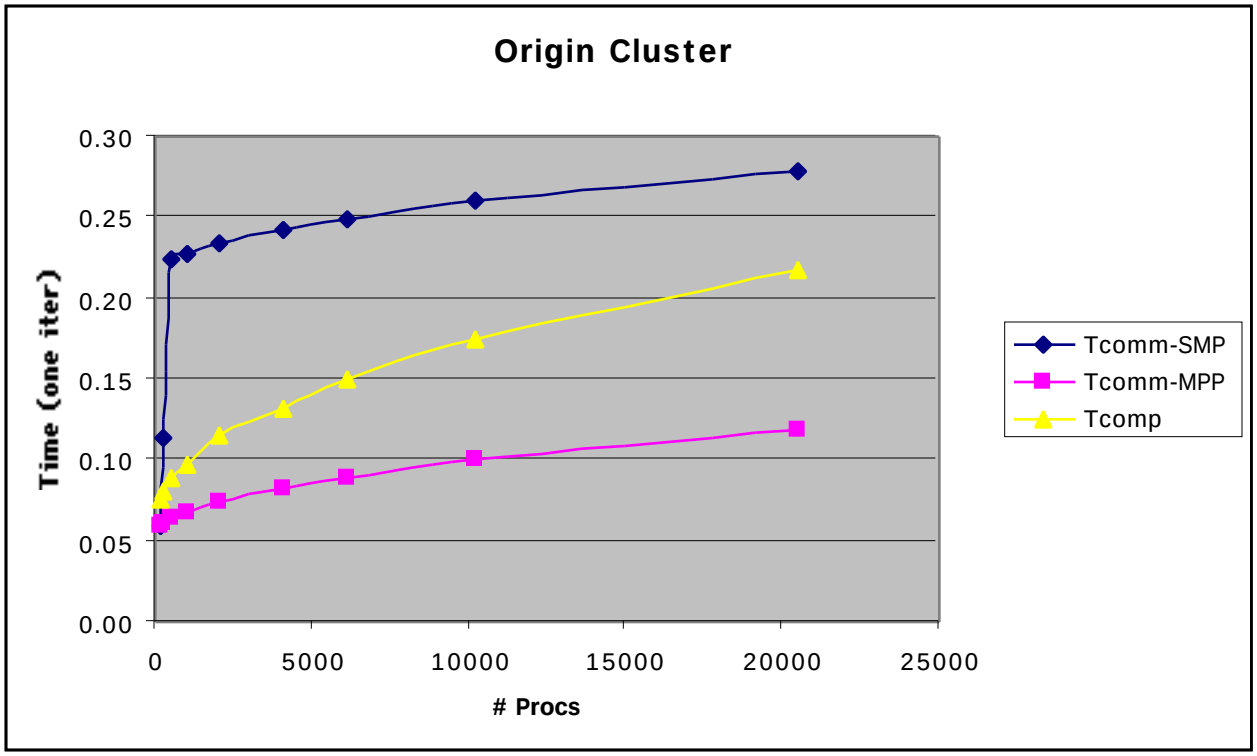
Application of Performance Models

- ***Thesis: the model is the "tool" used for: analysis, optimization, design, engineering and prediction. The "tools" as we know them, are mostly for data collection.***

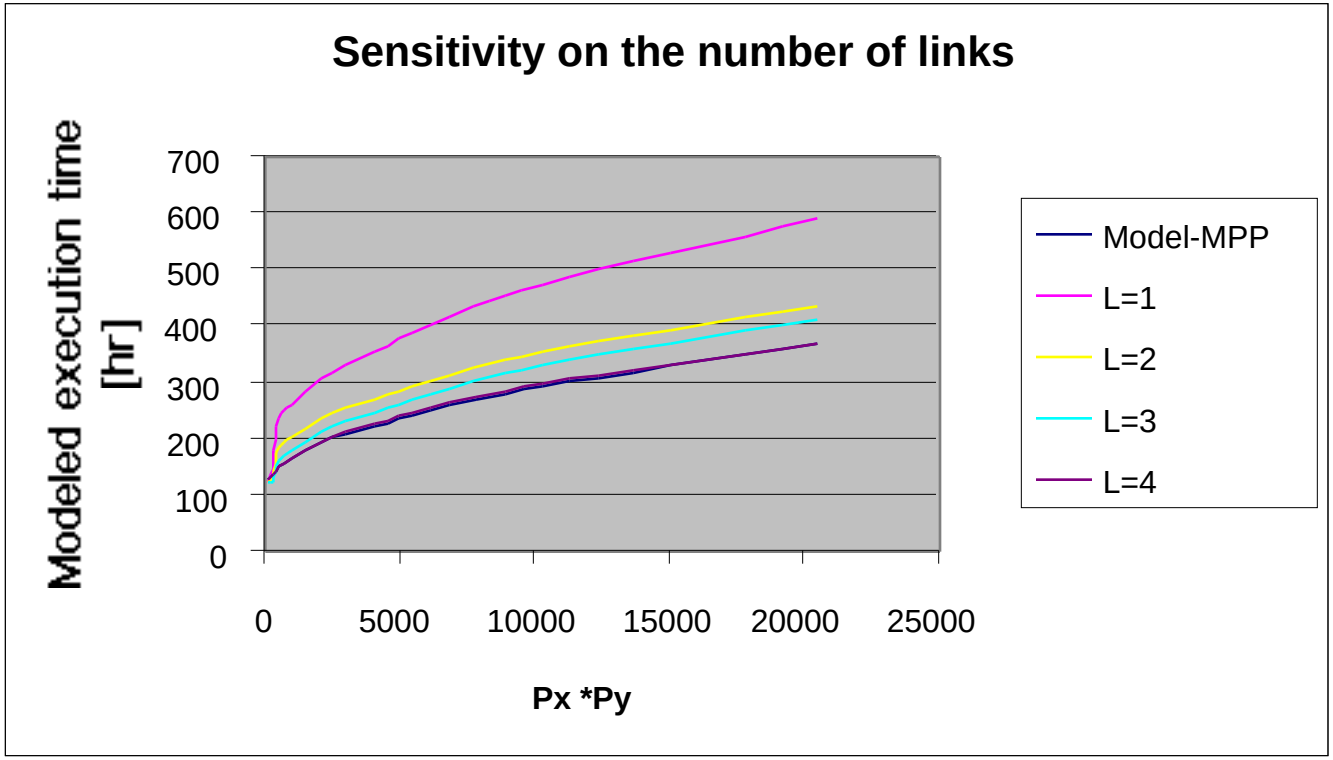
- Analysis
- Prediction
- Parallel Architecture Design
- Application Design and Implementation

ASCI WORKLOAD

- **Deterministic particle transport (structured grids)**
- **Hydro (with AMR)**
- **Deterministic particle transport (on unstructured grids)**
- **Non-deterministic particle transport**

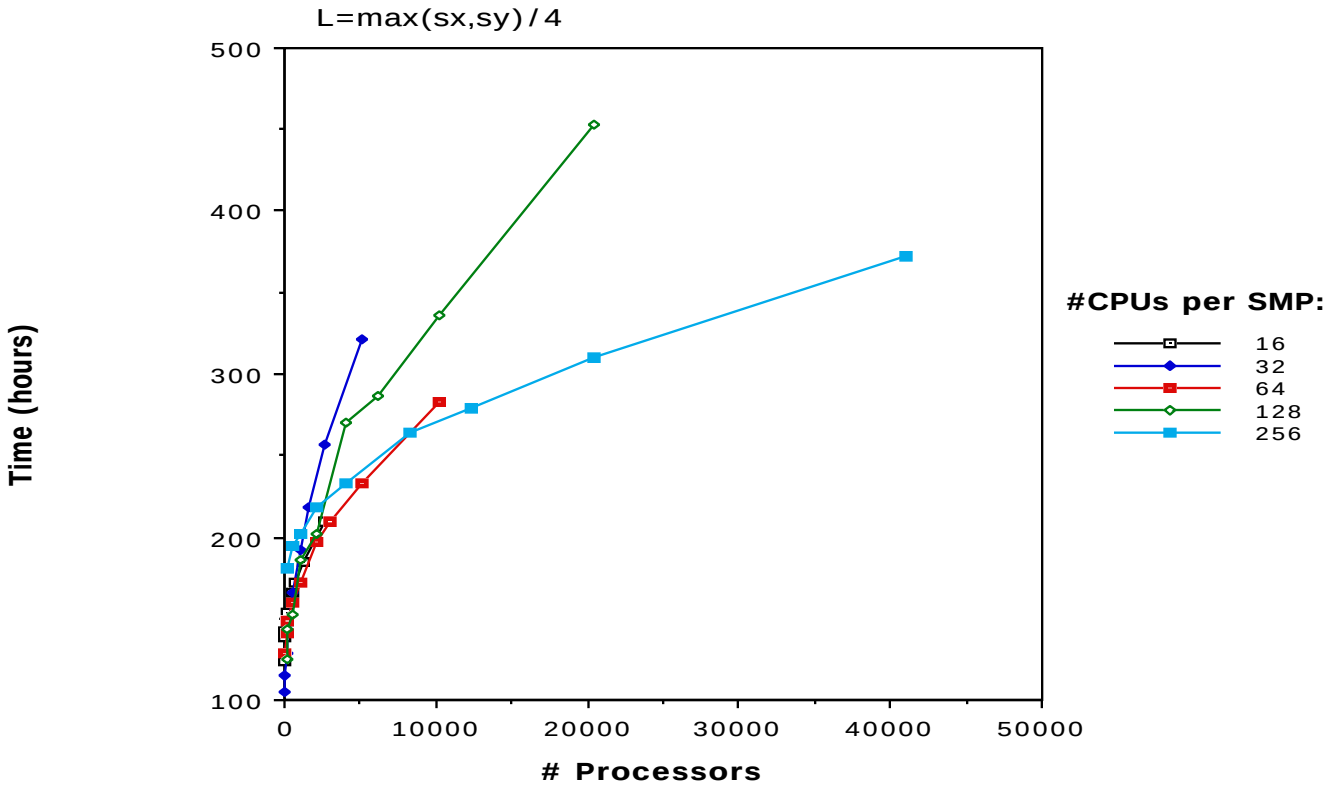


50 MFLOPS/cpu, Lat=100 us, BW=100MB/s,
4 x 4 x 100 subgrid, optimal blocking, 10e7 cells total,
1 Link ea. Dir. Between Hosts.

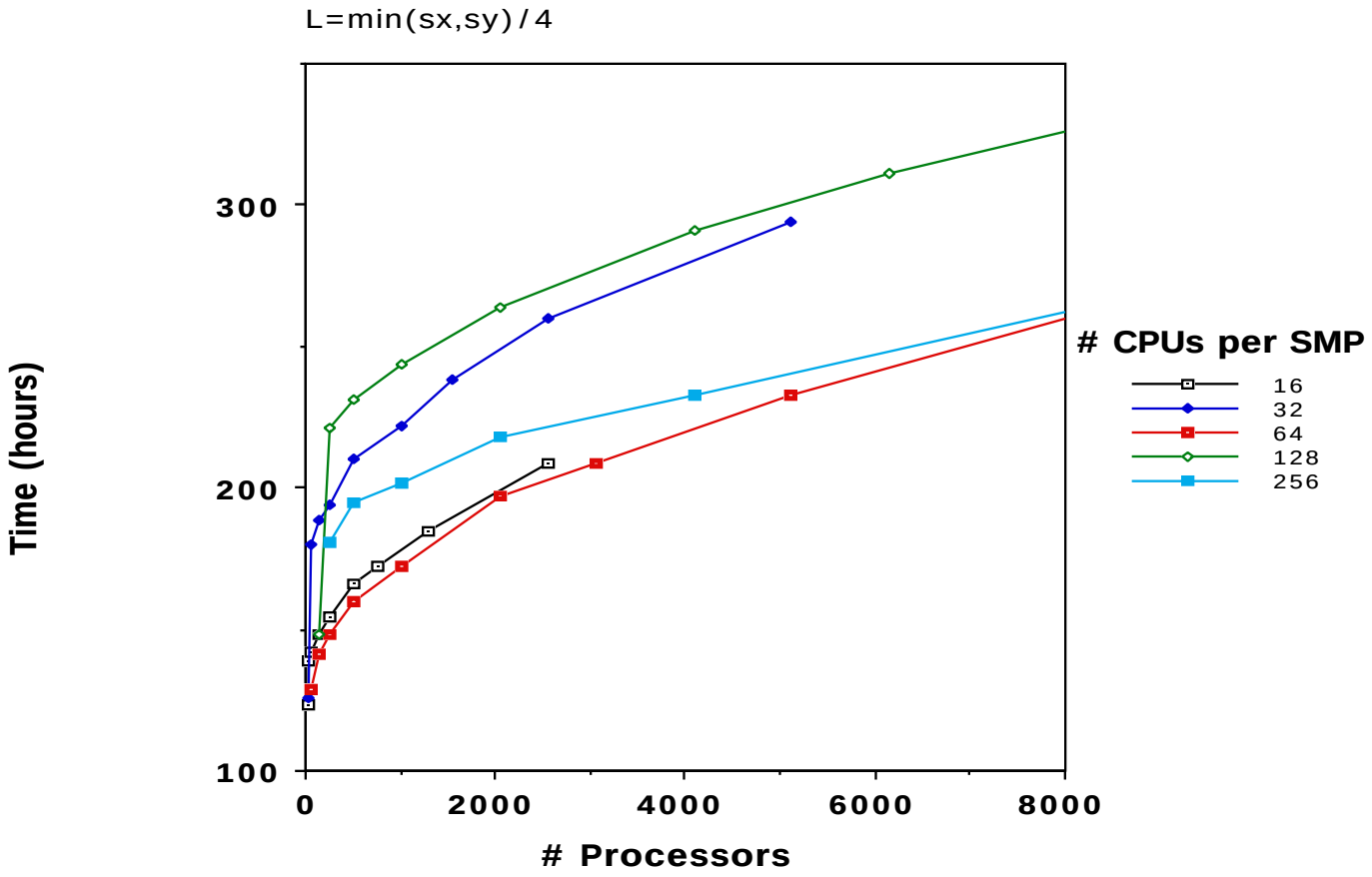


50 MFLOPS/cpu, Lat=100 us, BW=100MB/s,
4 x 4 x 100 subgrid, optimal blocking, 10e7 cells total,
NG=30,12 iters, 10e4 timesteps

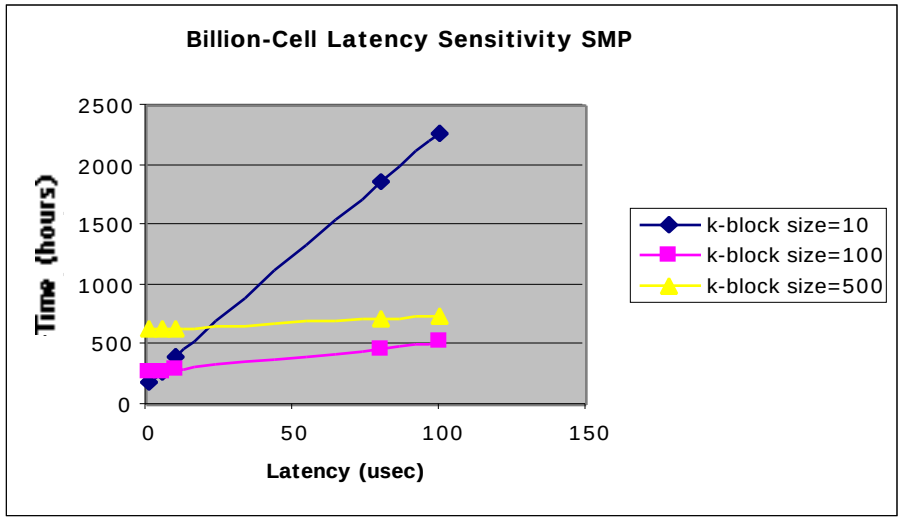
Sensitivity Analysis on SMP Size

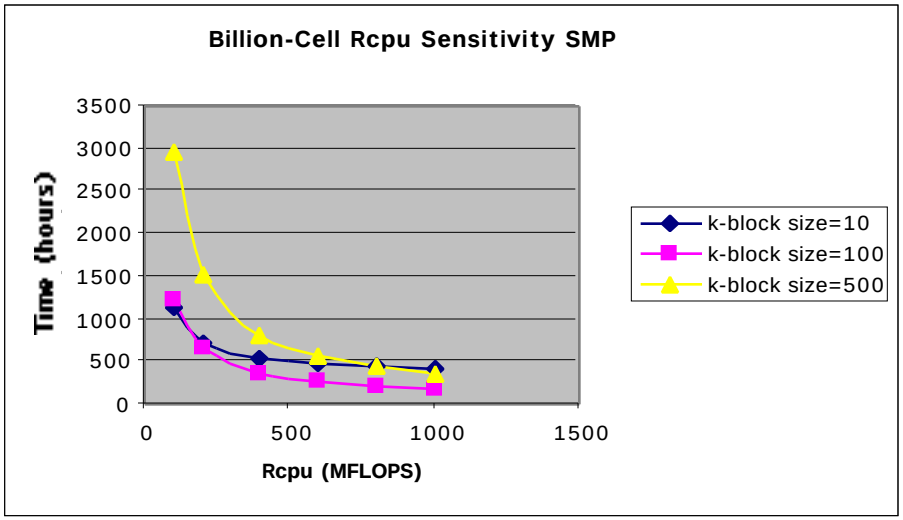


Sensitivity Analysis on SMP Size



Particle Transport Scalability Results: 1 Billion Cells on an SMP cluster

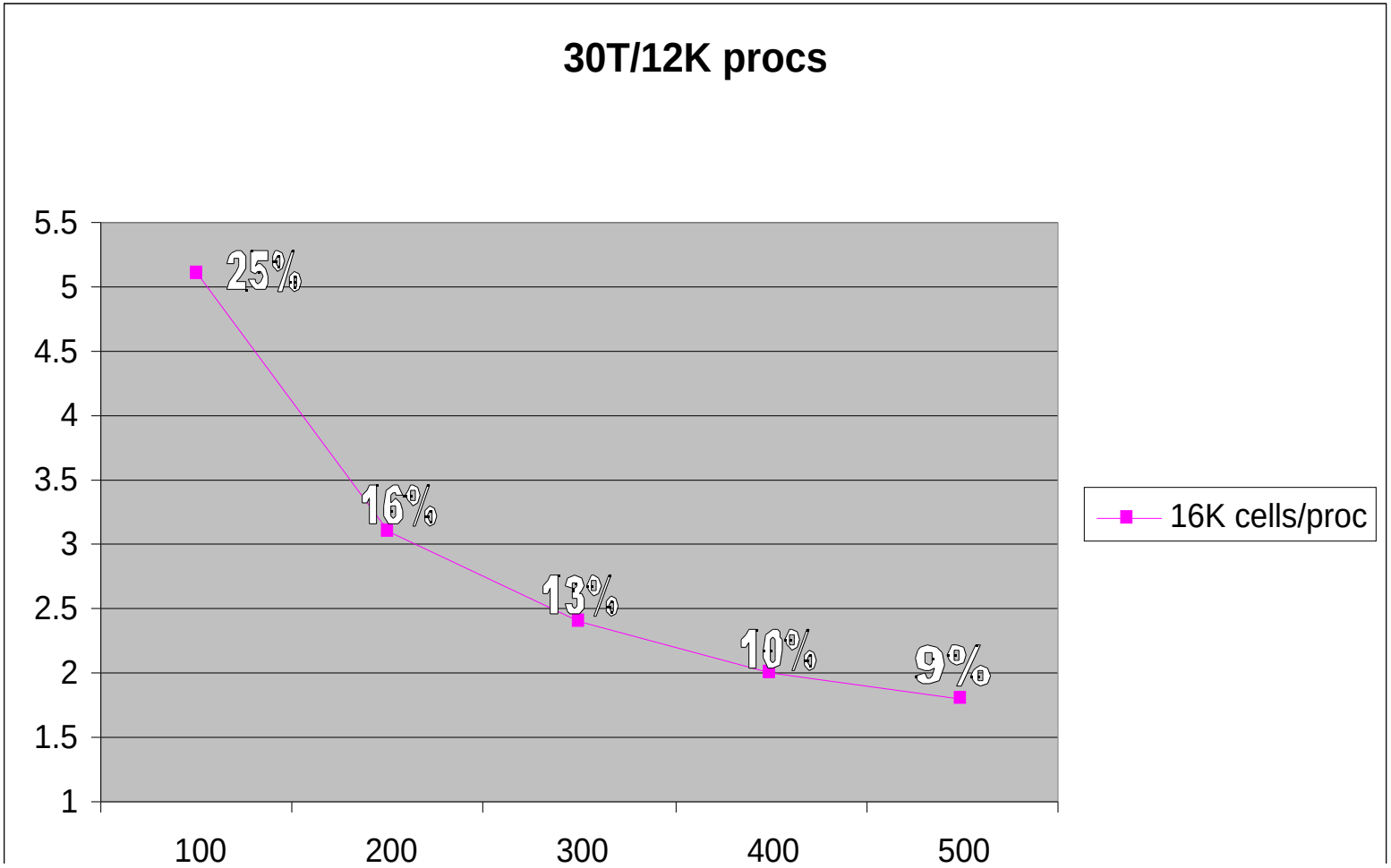




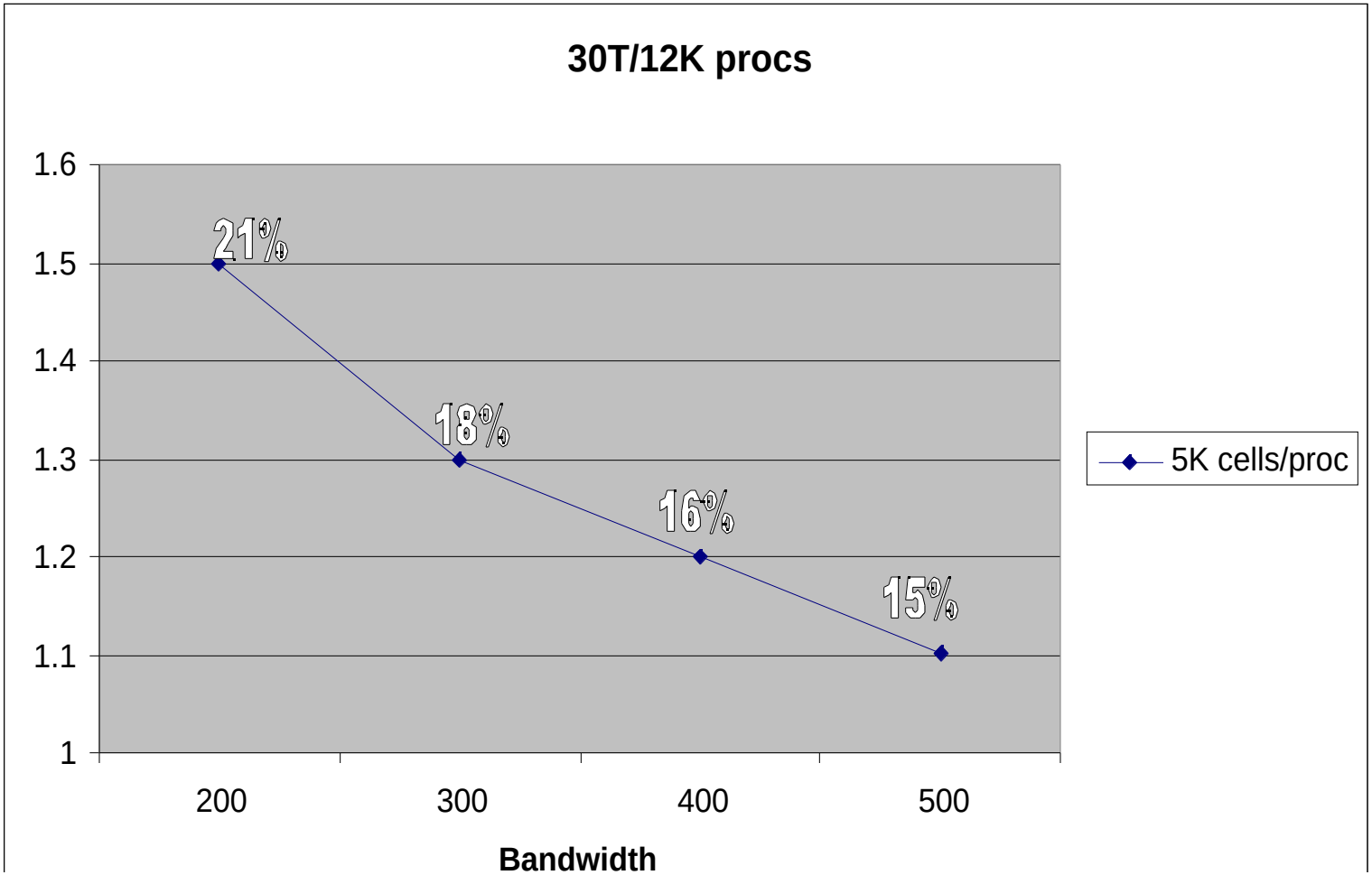
Estimates of SWEEP3D Performance on a Hypothetical Future-Generation (100-TFLOPS) System as a Function of MPI Latency and Sustained Per-Processor Computing Rate.

	Sustained Computing Rate	
	10% of Peak	*50% of Peak*
MPI Latency	Runtime (hours)	Runtime (hours)
0.1 μ_s	178	58
1.0 μ_s	205	68
10 μ_s	264	104

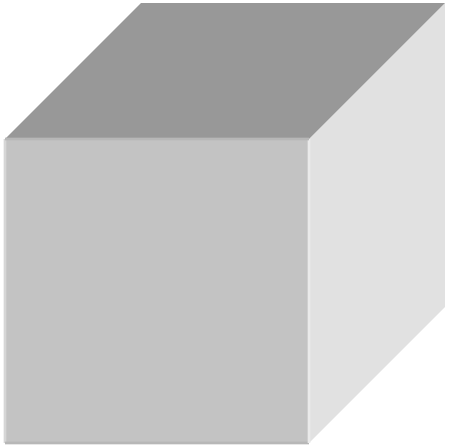
Applications of Modeling: SWEEP3D (I)



SWEEP3D (II)


30T/12K procs
1.6
1.5
1.4
1.3
1.2
1.1
1
21%
18%
16%
15%
200
300
400
500
Bandwidth
5K cells/proc

4) The Use
B) Algorithmic Changes



- 3-D cell Grid
- Partition in 3 dimensions,
  - each PE can have:

    i cells in X, j cells in Y, k cells in Z

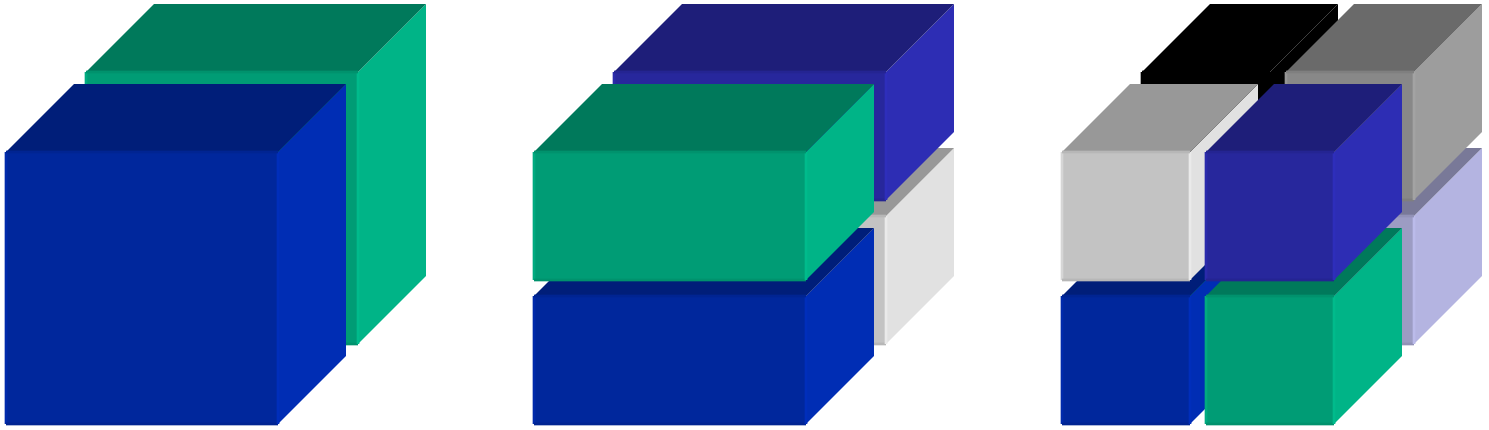
Volume = i.j.k (computation)

Surface = i.j + j.k + k.i (communication - gather/scatter

$$\frac{\text{Surface}}{\text{Volume}} = \frac{1}{i} + \frac{1}{j} + \frac{1}{k}$$

(min when i=j=k)

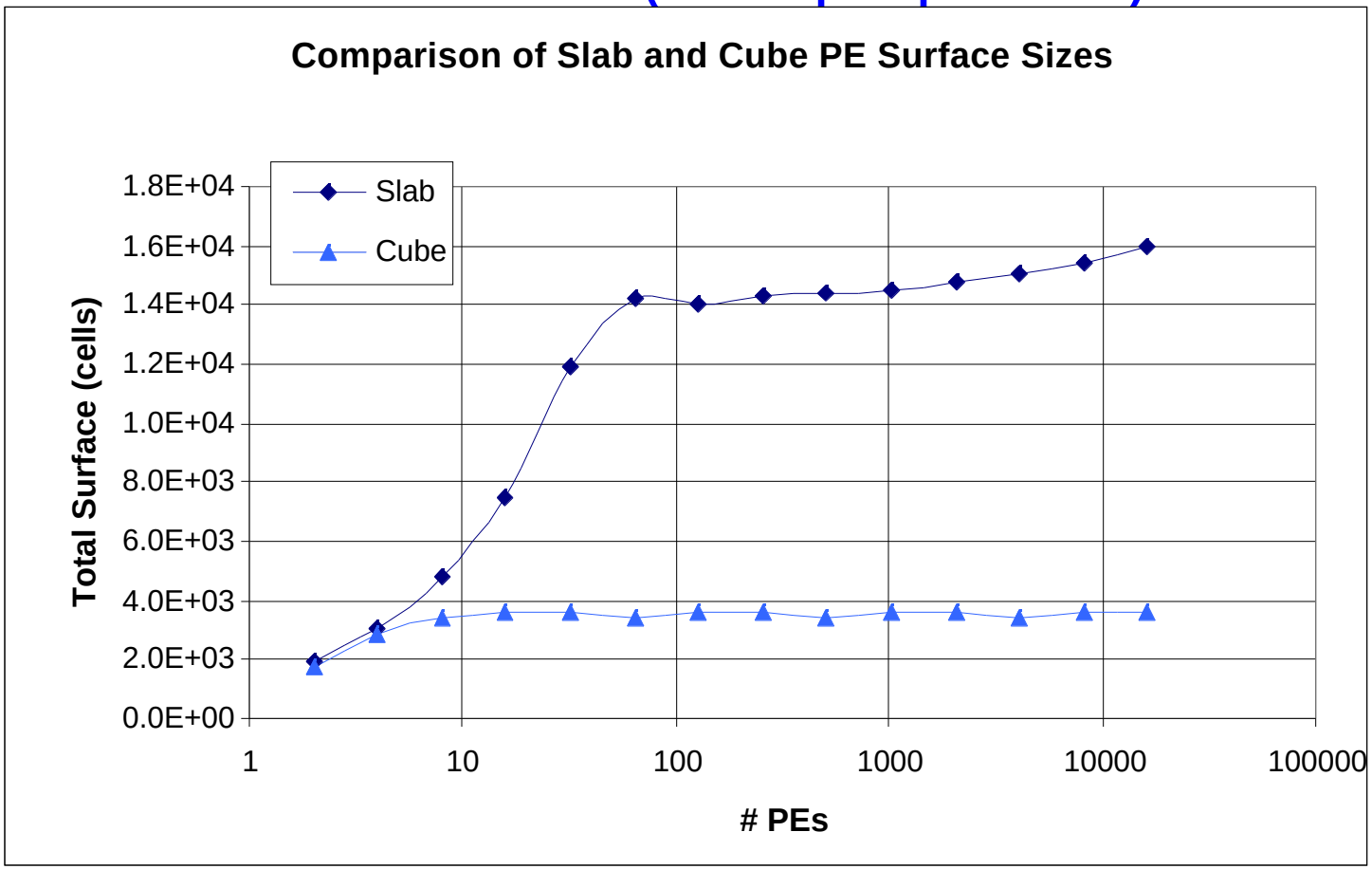
Cube Decomposition



Case 1: 2x2x1 **Case 2: 2x1x1** **Case 3: 1x1x1**

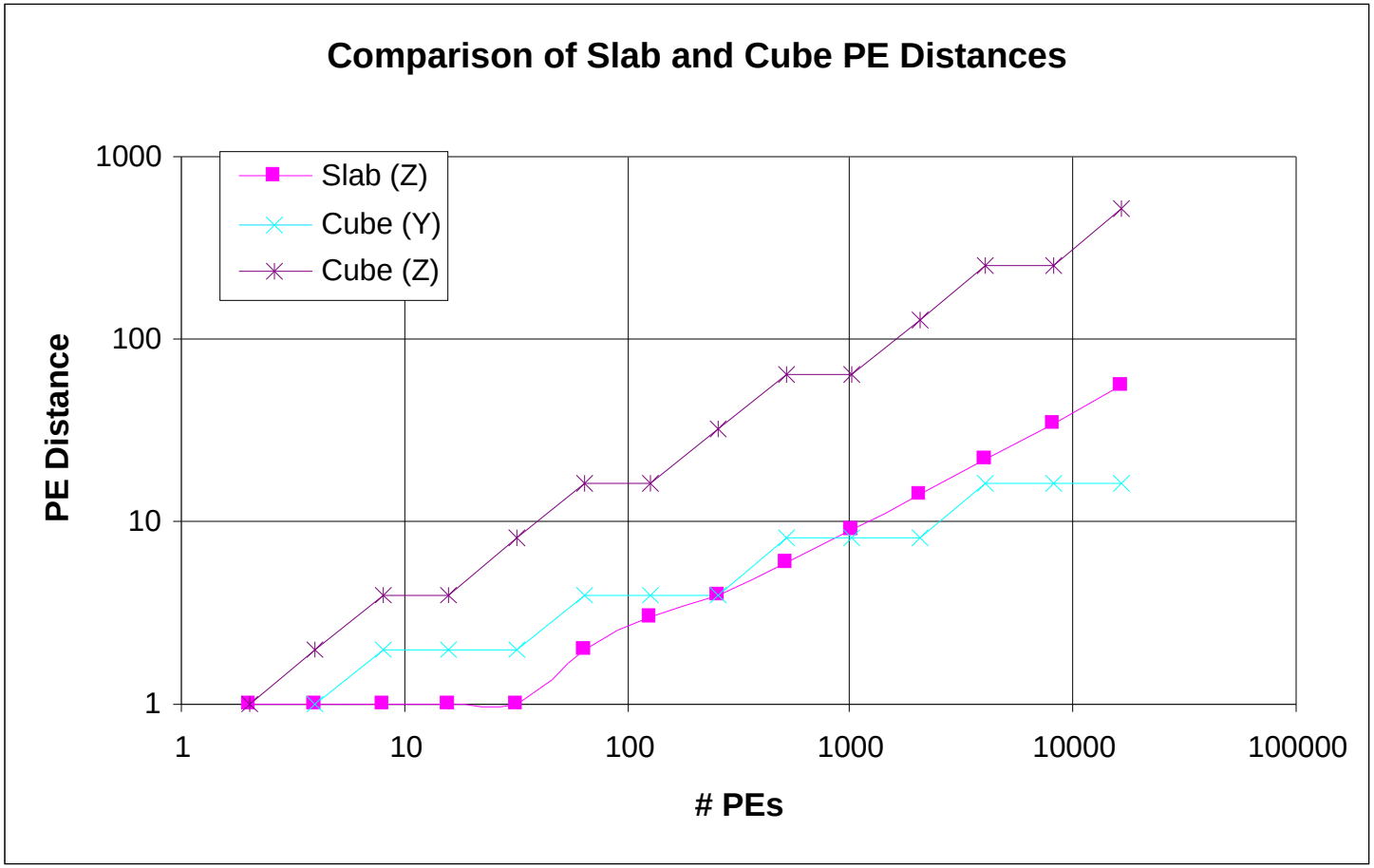
- Minimum Surface to Volume ratio
  - minimizes communication time (Gather & Scatter)

Cube vs Slab (Compaq ES45)



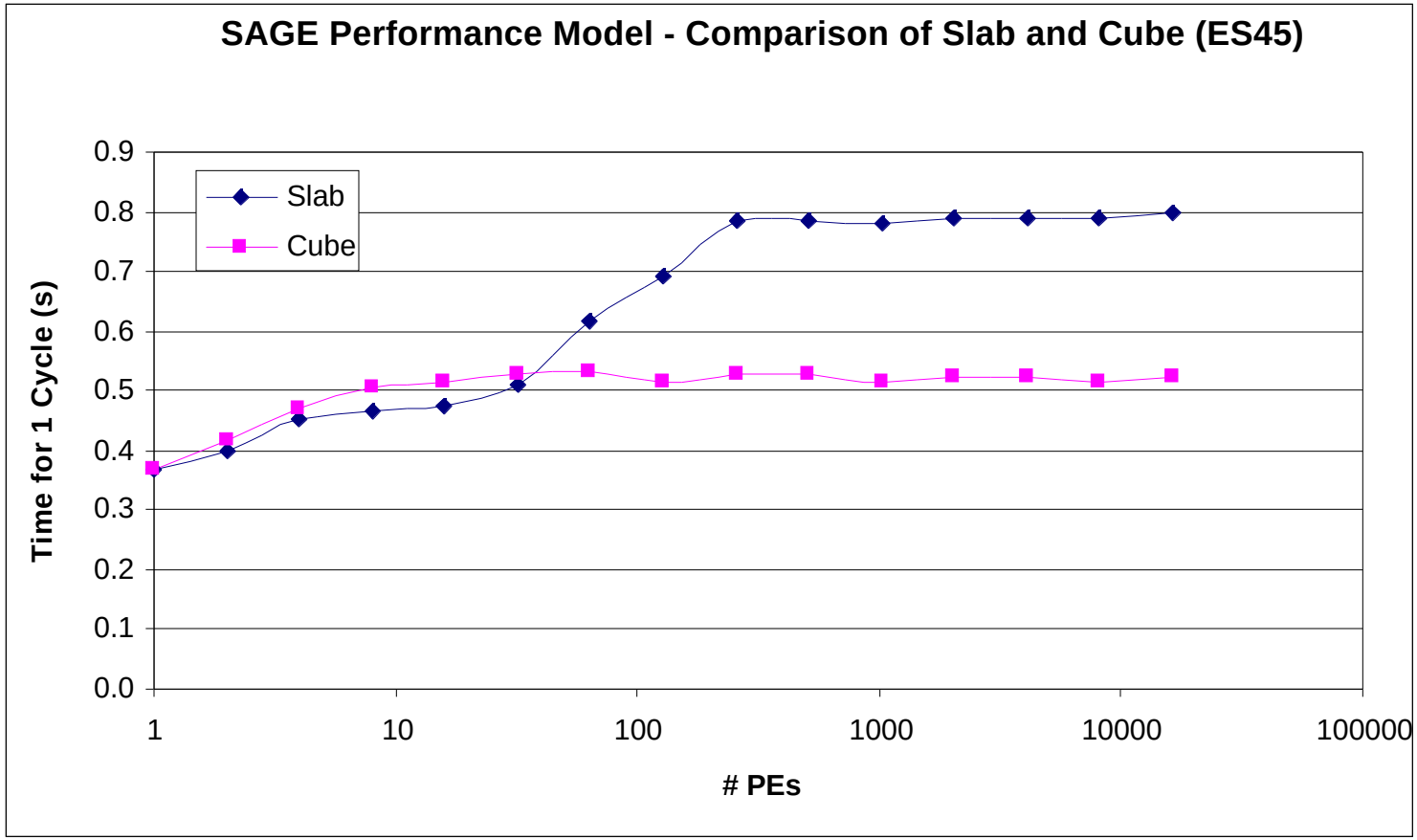
- Cube Surface 4 times less than Slab

Cube vs Slab (Compaq ES45)



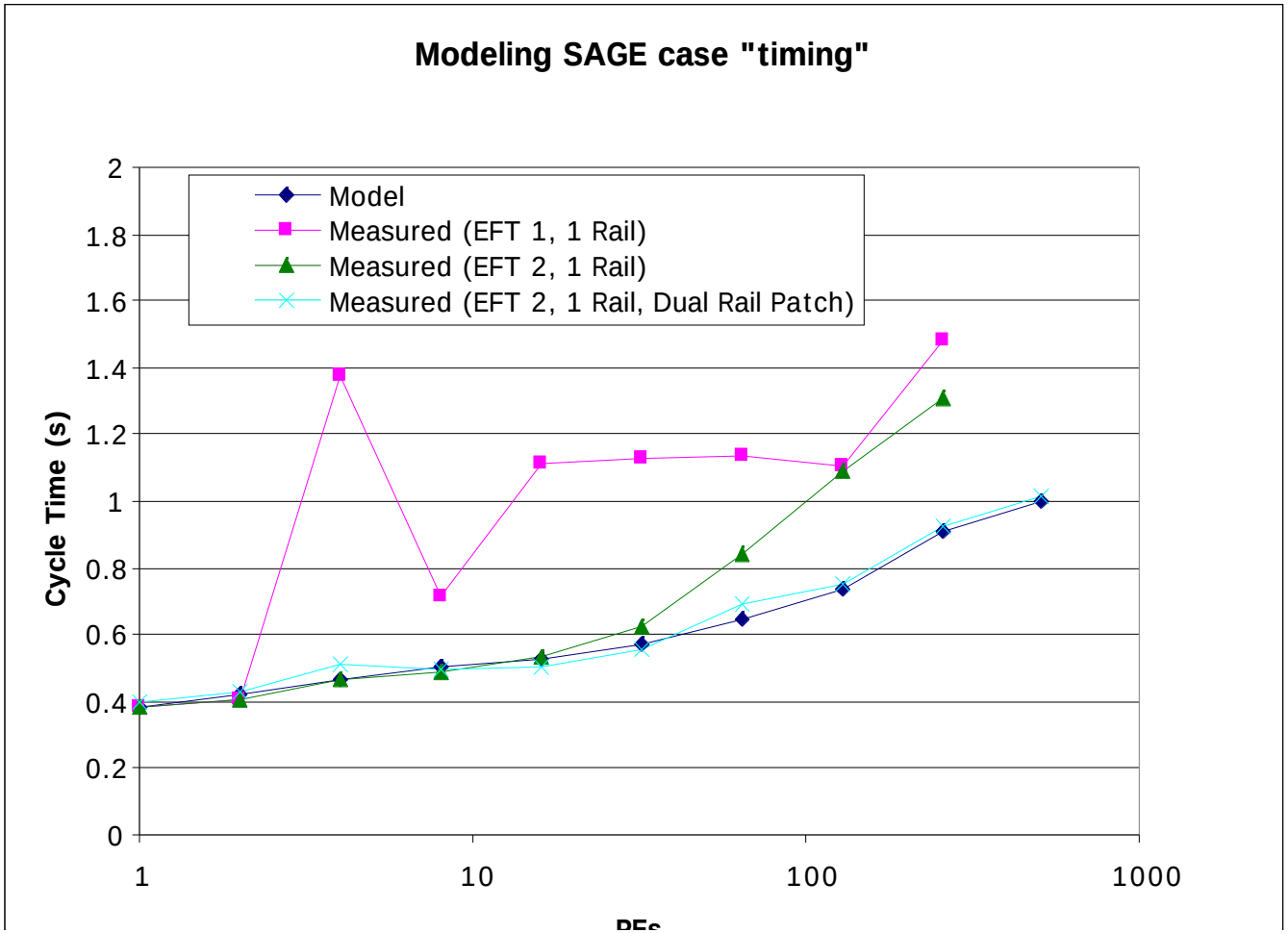
- Cube PE distance > Slab PE distance

Cube vs Slab (Compaq ES45)



- Expect - performance improvement using cube

System Diagnostics



Conclusions

- Application / architecture mapping is the key - not lists of raw basic machine characteristics.
- Point design studies need to address a specific workload.
- Performance and scalability modeling is an effective “tool” for workload characterization, system design, application optimization, and algorithm-architecture mapping.
- Back-of-the-envelope performance predictions are risky (outright wrong ?), given the complexity of analysis in a multidimensional performance space.
- Applications and systems at this scale need to be performance-engineered -- modeling is the means to analysis.

Resources and Acknowledgements

- An updated list of our publications in all the areas described,as well as descriptions of our projects can be found at:

http://www.c3.lanl.gov/cic3/teams/par_arch/Publications.html

- Parallel Architectures and Performance Team members and key contributors to the work presented:

Hank Alme, Salvador Coll, Eitan Frachtenberger, Adolfy Hoisie,
Darren Kerbyson, Juan Penador, Fabrizio Petrini, Harvey Wasserman